Search Algorithms for Software-Only Real-Time Recognition with Very Large Vocabularies

Long Nguyen, Richard Schwartz, Francis Kubala, Paul Placeway

BBN Systems & Technologies
70 Fawcett Street, Cambridge, MA 02138

ABSTRACT

This paper deals with search algorithms for real-time speech recognition. We argue that software-only speech recognition has several critical advantages over using special or parallel hardware. We present a history of several advances in search algorithms, which together, have made it possible to implement real-time recognition of large vocabularies on a single workstation without the need for any hardware accelerators. We discuss the Forward-Backward Search algorithm in detail, as this is the key algorithm that has made possible recognition of very large vocabularies in real-time. The result is that we can recognize continuous speech with a vocabulary of 20,000 words strictly in real-time entirely in software on a high-end workstation with large memory. We demonstrate that the computation needed grows as the cube root of the vocabulary size.

1. Introduction

The statistical approach to speech recognition requires that we compare the incoming speech signal to our model of speech and choose as our recognized sentence that word string that has the highest probability, given our acoustic models of speech and our statistical models of language. The required computation is fairly large. When we realized that we needed to include a model of understanding, our estimate of the computational requirement was increased, because we assumed that it was necessary for all of the knowledge sources in the speech recognition search to be tightly coupled.

Over the years DARPA has funded major programs in special-purpose VLSI and parallel computing environments specifically for speech recognition, because it was taken for granted that this was the only way that real-time speech recognition would be possible. However, these directions became major efforts in themselves. Using a small number of processors in parallel was easy, but efficient use of a large number of processors required a careful redesign of the recognition algorithms. By the time high efficiency was obtained, there were often faster uniprocessors available.

Design of special-purpose VLSI obviously requires considerable effort. Often by the time the design is completed, the algorithms implemented are obsolete and much faster general purpose processors are available in workstations. The result is that neither of these approaches has resulted in real-time recognition with vocabularies of 1,000 words or more.

Another approach to the speech recognition search problem is to reduce the computation needed by changing the search algorithm. For example, IBM has developed a flexible stack-based search algorithm and several *fast match* algorithms that reduce the search space by quickly eliminating a large fraction of the possible words at each point in the search. In 1989 we, at BBN [1], and others [2, 3] developed the N-best Paradigm, in which we use a powerful but inexpensive model for speech to find the top N sentence hypotheses for an utterance, and then we rescore each of these hypotheses with more complex models. The result was that the huge search space described by the complex models could be avoided, since the space was constrained to the list of N hypotheses. Even so, an exact algorithm for the N-best sentence hypotheses required about 100 times more computation than the simple Viterbi search for the most likely sentence.

In 1990 we realized that we could make faster advances in the algorithms using off-the-shelf hardware than by using special hardware. Since then we have gained orders of magnitude in speed in a short time by changing the search algorithms in some fundamental ways, without the need for additional or special hardware other than a workstation. This has resulted in a major paradigm shift. We no longer think in terms of special-purpose hardware – we take it for granted that recognition of any size problem will be possible with a software-only solution.

There are several obvious advantages to software-based recognizers: greater flexibility, lower cost, and the opportunity for large gains in speed due to clever search algorithms.

1. Since the algorithms are in a constant state of flux, any special-purpose hardware is obsolete before it is finished.

2. Software-only systems are key to making the technology broadly usable.
   - Many people will simply not purchase extra hardware.
   - Integration is much easier.

- The systems are more flexible.

3. For those people who already have workstations, software is obviously less expensive.
4. Most importantly, it is possible to obtain much larger gains in speed due to clever search algorithms than from faster hardware.

We have previously demonstrated real-time software-only recognition for the ATIS task with over 1,000 words. More recently, we have developed new search algorithms that perform recognition of 20,000 words with fully-connnected bigram and trigram statistical grammars in strict real-time with little loss in recognition accuracy relative to research levels.

First, we will very briefly review some of the search algorithms that we have developed. Then we will explain how the Forward-Backward Search can be used to achieve real-time 20,000-word continuous speech recognition.

2. Previous Algorithms

The two most commonly used algorithms for speech recognition search are the time-synchronous beam search [4] and the best-first stack search [5]. (We do not consider "island-driven" searches here, since they have not been shown to be effective.)

2.1. Time-Synchronous Search

In the time-synchronous Viterbi beam search, all the states of the model are updated in lock step frame-by-frame as the speech is processed. The computation required for this simple method is proportional to the number of states in the model and the number of frames in the input. If we discard any state whose score is far below the highest score in that frame we can reduce the computation by a large factor.

There are two important advantages of a time-synchronous search. First, it is necessary that the search be time-synchronous in order for the computation to be finished at the same time that the speech is finished. Second, since all of the hypotheses are of exactly the same length, it is possible to compare the scores of different hypotheses in order to discard most hypotheses. This technique is called the *beam search*. Even though the beam search is not theoretically admissible, it is very easy to make it arbitrarily close to optimal simply by increasing the size of the beam. The computational properties are fairly well-behaved with minor differences in speech quality.

One minor disadvantage of the Viterbi search is that it finds the state sequence with the highest probability rather than the word sequence with the highest probability. This is only a minor disadvantage because the most likely state sequence has been empirically shown to be highly correlated to the most likely word sequence. (We have shown in [6] that a slight modification to the Viterbi computation removes this problem, albeit with a slight approximation. When two paths come to the same state at the same time, we add the probabilities instead of taking the maximum.) A much more serious problem with the time-synchronous search is that it must follow a very large number of theories in parallel even though only one of them will end up scoring best. This can be viewed as wasted computation.

We get little benefit from using a fast match algorithm with the time-synchronous search because we consider starting all possible words at each frame. Thus, it would be necessary to run the fast match algorithm at each frame, which would be too expensive for all but the least expensive of fast match algorithms.

2.2. Best-First Stack Search

The true best-first search keeps a sorted stack of the highest scoring hypotheses. At each iteration, the hypothesis with the highest score is advanced by all possible next words, which results in more hypotheses on the stack. The best-first search has the advantage that it can theoretically minimize the number of hypotheses considered if there is a good function to predict which theory to follow next. In addition, it can take very good advantage of a fast match algorithm at the point where it advances the best hypothesis.

The main disadvantage is that there is no guarantee as to when the algorithm will finish, since it may keep backing up to shorter theories when it hits a part of the speech that doesn't match well. In addition it is very hard to compare theories of different length.

2.3. Pseudo Time-Synchronous Stack Search

A compromise between the strict time-synchronous search and the best-first stack search can be called the Pseudo Time-Synchronous Stack Search. In this search, the shortest hypothesis (i.e. the one that ends earliest in the signal) is updated first. Thus, all of the active hypotheses are within a short time delay of the end of the speech signal. To keep the algorithm from requiring exponential time, a beam-type pruning is applied to all of the hypotheses that end at the same time. Since this method advances one hypothesis at a time, it can take advantage of a powerful fast match algorithm. In addition, it is possible to use a higher order language model without the computation growing with the number of states in the language model.

2.4. N-best Paradigm

The N-best Paradigm was introduced in 1989 as a way to integrate speech recognition with natural language processing. Since then, we have found it to be useful for applying the more expensive speech knowledge sources as well, such as cross-word models, tied-mixture densities, and trigram language models. We also use it for parameter and weight optimization. The N-best Paradigm is a type of fast match at the sentence level. This reduces the search space to a short list of likely whole-sentence hypotheses.

The Exact N-best Algorithm [1] has the side benefit that it is also the only algorithm that guarantees finding the most likely sequence of words. Theoretically, the computation required for this algorithm cannot be proven to be less than exponential with the length of the utterance. However, this case only exists when all the models of all of the phonemes and words are identical (which would present a more serious problem than large computation). In practice, we find that the computation required can be made proportional to the number of hypotheses desired, by the use of techniques similar to the beam search.

Since the development of the exact algorithm, there have been several approximations developed that are much faster, with varying degrees of accuracy [2, 3, 7, 8]. The most recent algorithm [9] empirically retains the accuracy of the exact algorithm, while requiring little more computation than that of a simple 1-best search.

The N-best Paradigm has the potential problem that if a knowledge source is not used to find the N-best hypotheses, the answer that would ultimately have the highest score including this knowledge source may be missing from the top N hypotheses. This becomes more likely as the error rate becomes higher and the utterances become longer. We have found empirically that this problem does not occur for smaller vocabularies, but it does occur when we use vocabularies of 20,000 words and trigram language models in the rescoring pass.

This problem can be avoided by keeping the lattice of all sentence hypotheses generated by the algorithm, rather than enumerating independent sentence hypotheses. Then the lattice is treated as a grammar and used to rescore all the hypotheses with the more powerful knowledge sources [10].

2.5. Forward-Backward Search Paradigm

The Forward-Backward Search algorithm is a general paradigm in which we use some inexpensive approximate time-synchronous search in the forward direction to speed up a more complex search in the backwards direction. This algorithm generally results in two orders of magnitude speedup for the backward pass. Since it was the key mechanism that made it possible to perform recognition with a 20,000-word vocabulary in real time, we discuss it in more detail in the next section.

3. The Forward-Backward Search Algorithm

We developed the Forward-Backward Search (FBS) algorithm in 1986 as a way to greatly reduce the computation needed to search a large language model. While many sites have adopted this paradigm for computation of the N-best sentence hypotheses, we feel that its full use may not be fully understood. Therefore, we will discuss the use of the FBS at some length in this section.

The basic idea in the FBS is to perform a search in the forward direction to compute the probability of each word ending at each frame. Then, a second more expensive search in the backward direction can use these word-ending scores to speed up the computation immensely. If we multiply the forward score for a path by the backward score of another path ending at the same frame, we have an estimate of the total score for the combined path, given the entire utterance. In a sense, the forward search provides the ideal fast match for the backward pass, in that it gives a good estimate of the score for each of the words that can follow in the backward direction, including the effect of all of the remaining speech.

When we first introduced the FBS to speed up the N-best search algorithm, the model used in the forward and backward directions were identical. So the estimate of the backward scores provided by the forward pass were exact. This method has also been used in a best-first stack search [8], in which it is very effective, since the forward-backward score for any theory covers the whole utterance. The forward-backward score solves the primary problem with the best-first search, which is that different hypotheses don't span the same amount of speech.

However, the true power of this algorithm is revealed when we use different models in the forward and backward directions. For example, in the forward direction we can use approximate acoustic models with a bigram language model. Then, in the backward pass we can use detailed HMM models with a trigram language model. In this case, the forward scores still provide an excellent (although not exact) estimate of the ranking of different word end scores. Because both searches are time-synchronous, it does not matter that the forward and backward passes do not get the same score. (This is in contrast to a backward best-first or A* search, which depends on the forward scores being an accurate prediction of the actual scores that will result in the backward pass.)

In order to use these approximate scores, we need to mod-

ify the algorithm slightly. The forward scores are normalized relative to the highest forward score at that frame. (This happens automatically in the BYBLOS decoder, since we normalized the scores in each frame in order to prevent underflow.) We multiply the normalized forward score by the normalized backward score to produce a normalized forward-backward score. We can compare these normalized forward-backward scores to the normalized backward scores using the usual beam-type threshold. This causes us to consider more than one path in the backwards direction. The best path (word sequence) associated with each word end may not turn out to be the highest, but this does not matter, because the backward search will rescore all the allowed paths anyway.

We find that the backward pass can run about 1000 times faster than it would otherwise, with the same accuracy. For example, when using a vocabulary of 20,000 words a typical beam search that allows for only a small error rate due to pruning requires about 20 times real time. In contrast, we find that the backward pass runs at about 1/60 real time! This makes it fast enough so that it can be performed at the end of the utterance with a delay that is barely noticeable.

But the FBS also speeds up the forward pass indirectly! Since we know there will be a detailed backward search, we need not worry about the accuracy of the forward pass to some extent. This allows us the freedom to use powerful approximate methods to speed up the forward pass, even though they may not be as accurate as we would like for a final score.

4. Sublinear Computation

Fast match methods require much less computation for each word than a detailed match. But to reduce the computation for speech recognition significantly for very large vocabulary problems, we must change the computation from one that is linear with the vocabulary to one that is essentially independent of the vocabulary size.

4.1. Memory vs Speed Tradeoffs

One of the classical methods for saving computation is to trade increased memory for reduced computation. Now that memory is becoming large and inexpensive, there are several methods open to us. The most obvious is various forms of fast match. We propose one such memory-intensive fast match algorithm here. Many others could be developed.

Given an unknown word, we can make several orthogonal measures on the word to represent the acoustic realization of that word as a single point in a multi-dimensional space. If we quantize each dimension independently, we determine a single (quantized) cell in this space. We can associate information with this cell that gives us a precomputed estimate of the HMM score of each word. The computation is performed only once, and is therefore very small and independent of the size of the vocabulary. (Of course the precompilation of the scores of each of the words given a cell in the space can be large.) The precision of the fast match score is limited only by the amount of memory that we have, and our ability to represent the scores efficiently.

4.2. Computation vs Vocabulary Size

To learn how the computation of our real-time search algorithm grows with vocabulary size we measured the computation required at three different vocabulary sizes: 1,500 words, 5,000 words, and 20,000 words. The time required, as a fraction of real time, is shown plotted against the vocabulary size in Figure 1.

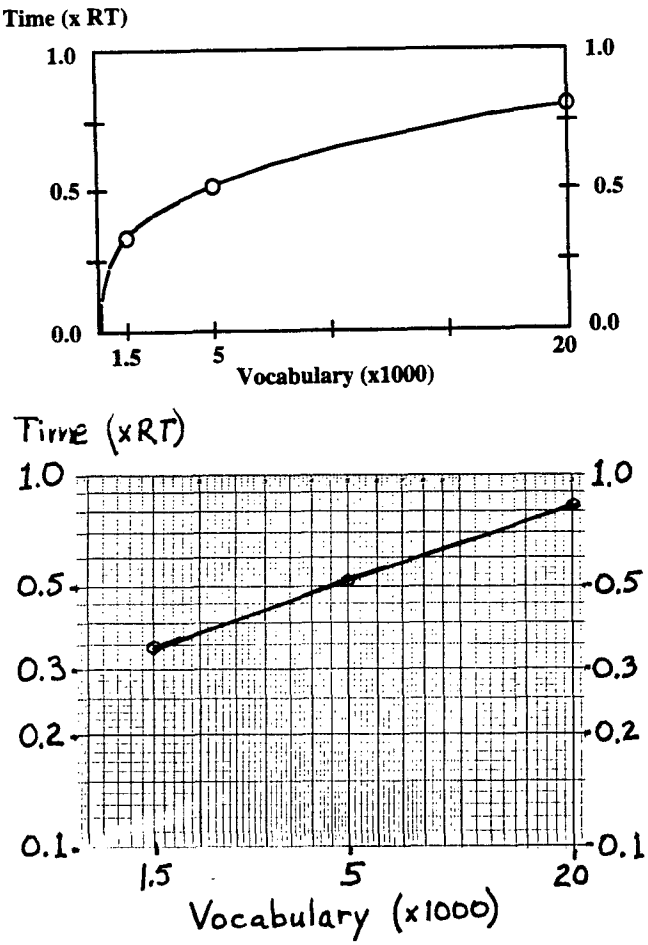

Figure 1: Run time vs vocabulary size. Plotted on a linear and a log-log scale.

As can be seen, the computation increases very slowly with increased vocabulary. To understand the behavior better we plotted the same numbers on a log-log scale as shown above.

Here we can see that the three points fall neatly on a straight line, leading us to the conclusion that the computation grows as a power of the vocabulary size, V. Solving the equation gives us the formula

$$time = 0.04 \quad V^{1/3} \tag{1}$$

This is very encouraging, since it means that if we can decrease the computation needed by a small factor it would be feasible to increase the vocabulary size by a much larger factor, making recognition with extremely large vocabularies possible.

5. Summary

We have discussed the search problem in speech recognition and concluded that, in our opinion, it is no longer worth considering parallel or special purpose hardware for the speech problem, because we have been able to make faster progress by modifying the basic search algorithm in software. At present, the fastest recognition systems are based entirely on software implementations. We reviewed several search algorithms briefly, and discussed the advantage of time-synchronous search algorithms over other basic strategies. The Forward-Backward Search algorithm has turned out to be an algorithm of major importance in that it has made possible the first real-time recognition of 20,000-word vocabularies in continuous speech. Finally, we demonstrated that the computation required by this algorithm grows as the cube root of the vocabulary size, which means that real-time recognition with extremely large vocabularies is feasible.

Acknowledgement

Some of this work was supported by the Defense Advanced Research Projects Agency and monitored by the Office of Naval Research under Contract Nos. N00014-91-C-0115, and N00014-92-C-0035.

References

1. Schwartz, R. and Y.L. Chow (1990) "The N-Best Algorithm: An Efficient and Exact Procedure for Finding the N Most Likely Sentence Hypotheses", ICASSP-90, April 1990, Albuquerque S2.12, pp. 81-84. Also in *Proceedings of the DARPA Speech and Natural Language Workshop*, Cape Cod, Oct. 1989.
2. V. Steinbiss (1989) "Sentence-Hypotheses Generation in a Continuous-Speech Recognition System," *Proc. of the European Conf. on Speech Communication and Technology, Paris, Sept. 1989, Vol. 2, pp. 51-54*
3. Mariño, J. and E. Monte (1989) "Generation of Multiple Hypothesis in Connected Phonetic-Unit Recognition by a Modified One-Stage Dynamic Programming Algorithm", *Proc. of the European Conf. on Speech Communication and Technology, Paris, Sept. 1989, Vol. 2, pp. 408-411*
4. Lowerre, B. (1977) "The Harpy Speech Recognition System", *Doctoral Thesis* CMU 1977.
5. Bahl, L.R., de Souza, P., Gopalakrishnan, P.S., Kanevsky, D., and D. Nahamoo (1990) "Constructing Groups of Acoustically Confusable Words". *Proceedings of the ICASSP 90*, April, 1990.
6. Schwartz, R.M., Chow, Y., Kimball, O., Roucos, S., Krasner, M., and J. Makhoul (1985) "Context-Dependent Modeling for Acoustic-Phonetic Recognition of Continuous Speech", *Proceedings of the ICASSP 85*, pp. 1205-1208, March, 1985.
7. R.M. Schwartz and S.A. Austin, "Efficient, High-Performance Algorithms for N-Best Search," *Proc. DARPA Speech and Natural Language Workshop*, Hidden Valley, PA, Morgan Kaufmann Publishers, pp. 6-11, June 1990.
8. Soong, F., Huang, E., "A Tree-Trellis Based Fast Search for Finding the N Best Sentence Hypotheses in Continuous Speech Recognition". *Proceedings of the DARPA Speech and Natural Language Workshop*, Hidden Valley, June 1990.
9. Alleva, F., Huang, X., Hwang, M-Y., Rosenfeld, R., "An Improved Search Algorithm Using Incremental Knowledge for Continuous Speech Recognition and An Overview of the SPHINX-II Speech Recognition System", *DARPA Human Language Technology Workshop*, Princeton, NJ, March, 1993.
10. Murveit, H., Butzberger, J., Digalakis, V., Weintraub, M., "Progressive-Search Algorithms for Large Vocabulary Speech Recognition", *DARPA Human Language Technology Workshop*, Princeton, NJ, March, 1993.