

---

# DYNAMIC PROGRAMMING SEARCH STRATEGIES: FROM DIGIT STRINGS TO LARGE VOCABULARY WORD GRAPHS

Hermann Ney<sup>1</sup> and Xavier Aubert<sup>2</sup>

<sup>1</sup> *Lehrstuhl für Informatik VI, RWTH Aachen –  
University of Technology, D-52056 Aachen, Germany*

<sup>2</sup> *Philips GmbH Forschungslaboratorien,  
D-52066 Aachen, Germany*

## ABSTRACT

This chapter gives an overview of the dynamic programming (DP) search strategy for large-vocabulary, continuous-speech recognition. Starting with the basic one-pass algorithm for word string recognition, we extend the search strategy to vocabularies of 20,000 words and more by using a tree organization of the vocabulary. Finally, we describe how this predecessor conditioned algorithm can be refined to produce high-quality word graphs. This method has been tested successfully on the Wall Street Journal task (American English, continuous speech, 20,000 words, speaker independent).

## 1 INTRODUCTION

Search strategies based on dynamic programming (DP) have been used successfully in a large number of speech recognition tasks, ranging from digit string recognition over 1,000-word recognition using heavily constrained grammars to 20,000-word unconstrained speech recognition.

Dynamic programming had already been used in the early days of automatic speech recognition [29, 3, 23]. Despite these successes, until recently, it was a highly controversial issue whether high-perplexity natural language speech recognition could be handled efficiently by dynamic programming. In 1992,

when we submitted a paper on this topic to a well reputed journal, we were advised by one of the reviewers that “current wisdom seems to indicate that a stack decoding ( $A^*$ ) strategy may be more efficient for large vocabulary recognition”. Meanwhile, three years later, even for 20,000-word tasks like the Wall Street Journal task, dynamic programming in one variant or the other has established itself as the most widely used search technique.

The characteristic features of the presentation given in this chapter are:

- The search problem in continuous speech recognition is put into the framework of statistical decision theory.
- As baseline strategy, we review the time-synchronous one-pass algorithm.
- The time-synchronous concept is extended towards a tree organization of the pronunciation lexicon so that the search effort is significantly reduced.
- As an alternative to the single-best concept, i.e. computing only the single most likely word sentence, we extend the one-pass strategy to produce a word graph or word lattice. We show that this word graph concept can be embedded in the successful one-pass beam search strategy that uses predecessor conditioned lexical trees. Basically only the bookkeeping technique has to be modified to employ the predecessor conditioned one-pass algorithm for word graph construction.
- For the word graph construction, we present recent experimental results for the 20,000-word Wall Street Journal (WSJ) task.

## 2 SYSTEM ARCHITECTURE

### 2.1 Bayes Decision Rule

Every approach to automatic speech recognition faces the problem of taking decisions in the presence of ambiguity and context and of modeling the interdependence of these decisions at various levels. If it were possible to recognize phonemes (or words) with a very high reliability, it would not be necessary to rely heavily on delayed decision techniques, error correcting techniques and statistical methods. In the near future, this problem of reliable and virtually error free phoneme or word recognition without using high-level knowledge is unlikely to be solved for large-vocabulary continuous-speech

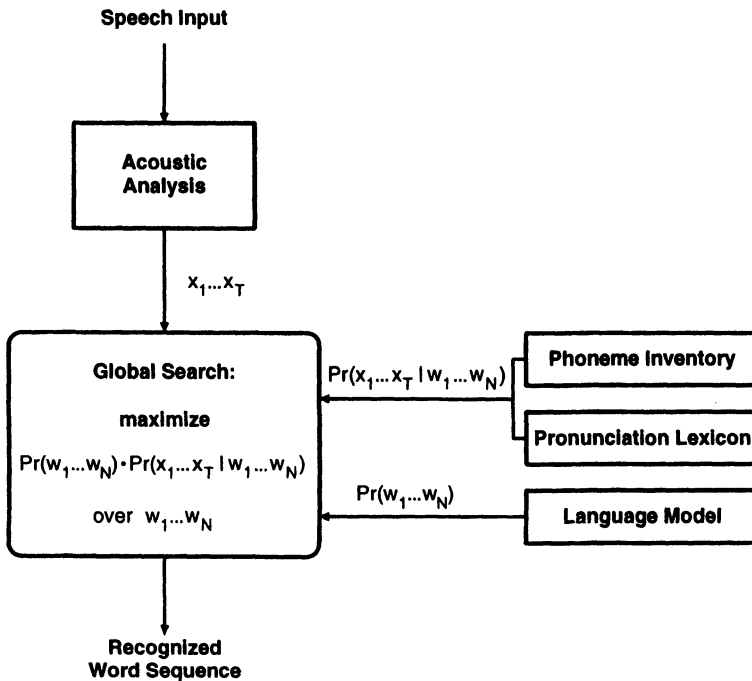


Figure 1 Bayes decision rule for speech recognition.

recognition. As a consequence, the recognition system has to deal with a large number of hypotheses about phonemes, words and sentences, and ideally has to take into account the “high-level constraints” as given by syntax, semantics and pragmatics. Given this state of affairs, statistical decision theory tells us how to minimize the probability of recognition errors [2]:

Maximize the posterior probability  $Pr(w_1...w_N|x_1...x_T)$ , i.e. determine the sequence of words  $w_1...w_n...w_N$  (of unknown length  $N$ ) which has most probably caused the observed sequence of acoustic vectors  $x_1...x_t...x_T$  (over time  $t = 1...T$ ), which are derived from the speech signal in the preprocessing step of acoustic analysis.

By applying Bayes’ theorem on conditional probabilities, the problem can be written in the following form: Determine the sequence of words  $w_1...w_n...w_N$

which maximizes

$$Pr(w_1...w_n...w_N) \cdot Pr(x_1...x_t...x_T|w_1...w_n...w_N) . \quad (1)$$

This so-called Bayes decision rule is illustrated in Fig. 1. It requires two types of probability distributions which we refer to as stochastic knowledge sources along with a search strategy:

- The language model, i.e.  $Pr(w_1...w_N)$ , is independent of the acoustic observations and is meant to incorporate restrictions on how to concatenate words of the vocabulary to form whole sentences.
- The acoustic-phonetic model, i.e.  $Pr(x_1...x_T|w_1...w_N)$ , is the conditional probability of observing the acoustic vectors  $x_1...x_T$  when the speaker utters the words  $w_1...w_N$ . Like the language model probabilities, these probabilities are estimated during the training phase of the recognition system. For a large-vocabulary system, there is typically a set of basic recognition units that are smaller than whole words. Examples of these so-called subword units are phonemes, demisyllables or syllables. The word models are then obtained by concatenating the subword models according to the phonetic transcription of the words in a pronunciation lexicon or dictionary.
- The decision on the spoken words must be taken by an optimization procedure which combines information of several knowledge sources: the language model, the acoustic-phonetic models of single phonemes, and the pronunciation lexicon. The optimization procedure is usually referred to as search in a state space defined by the knowledge sources.

## 2.2 Stochastic Knowledge Sources

The task of a language model is to capture the restrictions on the combinations of words due to the inherent redundancy of the language subset handled by the system. This redundancy results from the syntactic, semantic and pragmatic constraints of the recognition task and may be modeled by probabilistic or nonprobabilistic ('yes/no') methods. In large vocabulary recognition tasks, the language model probabilities are typically approximated by bigram or trigram models:

$$Pr(w_n|w_1...w_{n-1}) \cong p(w_n|w_{n-1}) \quad (2)$$

$$Pr(w_n|w_1...w_{n-1}) \cong p(w_n|w_{n-2}, w_{n-1}) . \quad (3)$$

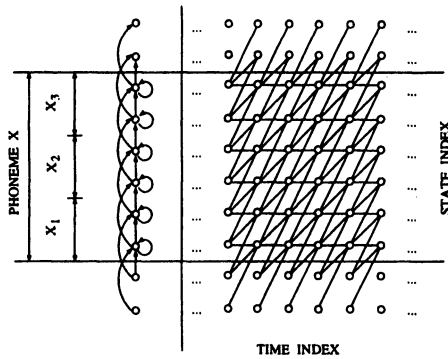


**Figure 2** Structure of a phoneme model and search space.

In order to reduce the number of parameters, bigram and trigram models can be defined at the level of word categories rather than words themselves [22].

As pointed out in the preceding section, the statistical approach requires the conditional probability  $Pr(x_1 \dots x_T | w_1 \dots w_N)$  of observing an acoustic vector sequence  $x_1 \dots x_T$ , given the word sequence  $w_1 \dots w_N$ . These probabilities are obtained by concatenating the corresponding word models, which again are obtained by concatenating phoneme or other subword unit models according to the pronunciation lexicon. As in many other systems, these subword units are modeled by so-called Hidden Markov Models (HMM). Hidden Markov Models are stochastic finite-state automata (or stochastic regular grammars) which consist of a Markov chain of acoustic states, modeling the temporal structure of speech, and a probabilistic function for each of the states, modeling the emission and observation of acoustic vectors [3, 2, 16]. A typical model along with the resulting search space for phonemes is shown in Fig. 2; here, the phoneme  $X$  consists of three parts ( $X_1, X_2, X_3$ ), resulting in a linear arrangement of six states. Words are obtained by concatenating the HMM phoneme units according to the nominal phonetic transcription. Usually, for a given state  $\sigma$  in a word model  $w$ , we have a transition probability  $a(s|\sigma; w)$  for going to state  $s$  and an emission probability (density)  $b(x_t|\sigma; w)$  for observing vector  $x_t$ . For the following, it suffices to consider only the product of the emission and transition probabilities:

$$q(x_t, s|\sigma; w) = a(s|\sigma; w) \cdot b(x_t|\sigma; w), \quad (4)$$

which is the conditional probability that, given state  $\sigma$  in word  $w$ , the acoustic vector  $x_t$  is observed and the state  $s$  is reached.

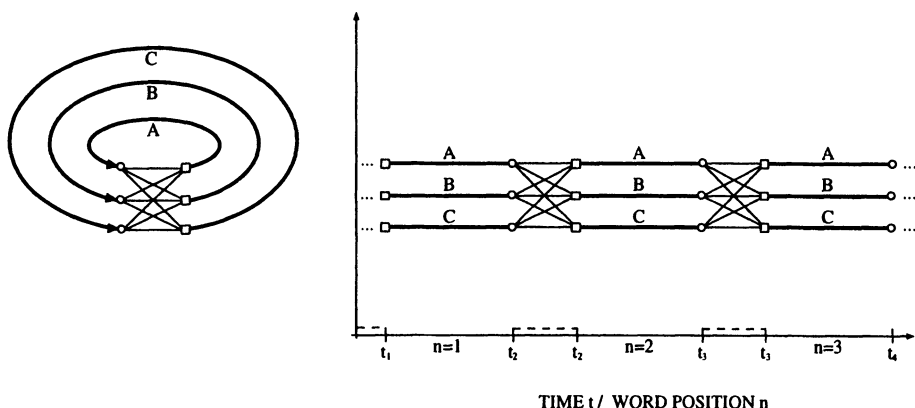


**Figure 3** Illustration of the search problem for a three-word vocabulary ( $A, B, C$ ).

### 3 BASELINE SEARCH: ONE-PASS ALGORITHM

#### 3.1 Specification of the Search Problem

In this section, we describe the one-pass algorithm which forms the baseline for all search strategies described in this chapter. Originally the one-pass algorithm had been designed for small vocabulary recognition tasks like digit string recognition. Over the last 20 years, however, it has turned out to be surprisingly successful in handling vocabularies of 10,000 and more words. The characteristic property of all search variants to be presented in this chapter is that they are conceptually based on a strictly time-synchronous left-to-right search strategy.

The decision on the spoken sentence is taken in the search procedure which attempts to determine the word sequence which best explains the input speech signal in terms of the given knowledge sources. The search space can be described as a huge finite-state machine [22, 20], which is defined in terms of the acoustic word models and the language model. For a three-word vocabulary, the search space is illustrated in Fig. 3. There are two types of transitions, namely the acoustic transitions representing the probabilities of the acoustic word models ( $A, B, C$  in Fig. 3) and the language transitions representing the language model probabilities. In Fig. 3, a bigram language model is assumed.

For each word bigram  $(v, w)$ , there is a language transition that is assigned the probability  $p(w|v)$  and that links the end of predecessor  $v$  to the beginning of word  $w$ . For recognition, as shown in Fig. 3, we unfold the finite-state machine along the time axis of the spoken utterance. For simplification reasons, Fig. 3 does not cover the details of the acoustic models and shows the language model transitions at times  $t_2$  and  $t_3$ , only. Both the acoustic transitions (as shown in Fig. 2) and the language transitions must be considered every 10-ms time frame. All in all, there is a huge number of possible state sequences, and all combinations of state and time must be systematically considered for recognition. In formulae, we have:

$$[w_1^N]_{opt} = \arg \max_{[w_1^N]} \left\{ Pr(w_1^N) \cdot \sum_{[s_1^T]} Pr(x_1^T; s_1^T | w_1^N) \right\}, \quad (5)$$

where the sum is to be taken over all paths  $[s_1^T]$  that are consistent with the word sequence  $[w_1^N]$  under consideration. Approximating the sum by the maximum, we obtain what is usually called Viterbi approximation:

$$[w_1^N]_{opt} = \arg \max_{[w_1^N]} \left\{ Pr(w_1^N) \cdot \max_{[s_1^T]} Pr(x_1^T; s_1^T | w_1^N) \right\}. \quad (6)$$

In other words, the 'most likely word sequence' is approximated by the 'most likely state sequence' [3, 2, 16]. Note that for the maximum approximation to work, we really need only the assumption that the resulting optimum word sequences are the same, not really that the maximum provides a good approximation to the sum.

In the maximum approximation, the search problem can be specified as follows. We wish to assign each acoustic vector at time  $t$  to a (state, word) index pair. This mapping can be viewed as a time alignment path, which is a sequence of (state, word) index pairs (stretching notation):

$$(s_1, w_1), \dots, (s_t, w_t), \dots, (s_T, w_T). \quad (7)$$

An example of such a time alignment path in connected word recognition is depicted in Fig. 4. For such paths, there are obvious continuity constraints or transition rules as shown in Fig. 5. Since the word models are obtained by concatenating phoneme models, the transition rules in the word interior (Fig. 5 top) are those of the used HMMs as shown in Fig. 2. At word boundaries (Fig. 5 bottom), we have to allow for transitions that link the terminal state  $S(v)$  of any predecessor word  $v$  to the beginning states  $s = 1$  and  $s = 2$  of any

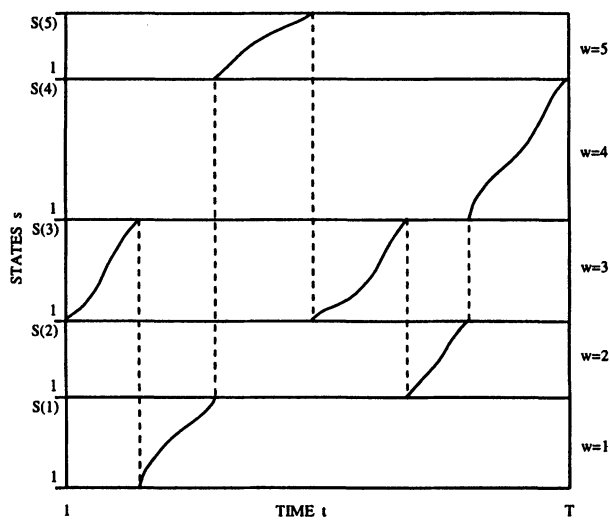


Figure 4 Example of a time alignment path.

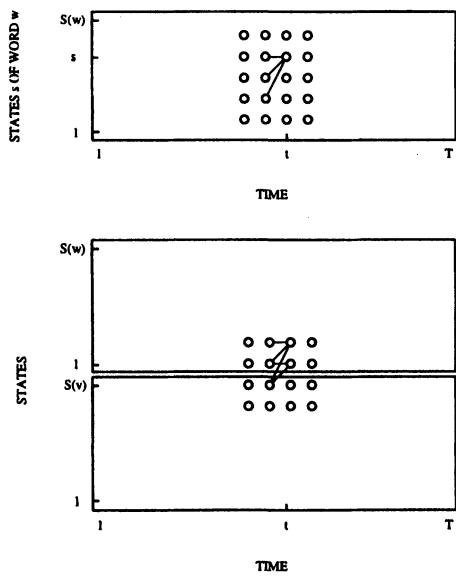


Figure 5 Path recombinations.

word  $w$ . The dynamic programming search to be presented will allow us to compute the probabilities

$$Pr(w_1 \dots w_N) \cdot Pr(x_1 \dots x_T; s_1 \dots s_T | w_1 \dots w_N) \quad (8)$$

in a left-to-right fashion over time  $t$  and to carry out the optimization over the unknown word sequence at the same time. Note that, unlike Eq. (5), the unknown word sequence and the unknown state sequence are determined simultaneously. Within the framework of the Viterbi criterion, the dynamic programming algorithm presents a closed-form solution for handling the interdependence of nonlinear time alignment, word boundary detection and word identification in continuous speech recognition [29, 23, 5, 19, 15].

### 3.2 Time-Synchronous Beam Search

The key concept of the dynamic programming strategy is based on the following two quantities:

$Q(t, s; w) :=$  score of the best path up to time  $t$  that ends in state  $s$  of word  $w$ .

$B(t, s; w) :=$  starting time of the best path up to time  $t$  that ends in state  $s$  of word  $w$ .

Although the back pointer  $B(t, s; w)$  is not needed for small-vocabulary tasks like digit string recognition, it is essential for vocabularies of 10,000 words and more to drastically reduce the storage requirements.

As shown in Fig. 5, there are two types of transition rules for the path, namely rules in the word interior and at word boundaries. The concept of dynamic programming is to use these rules to decompose the path into two parts and formulate recurrence relations, which can be solved by filling in tables, namely  $Q(t, s; w)$  for the problem under consideration. In a more general setting of optimization problems, this concept is often referred to as *Bellman's principle of optimality* [4]. In the word interior, we have the recurrence equation:

$$Q(t, s; w) = \max_{\sigma} \{ q(x_t, s | \sigma; w) \cdot Q(t-1, \sigma; w) \} \quad (9)$$

$$B(t, s; w) = B(t-1, \sigma_{\max}(t, s; w); w), \quad (10)$$

where  $\sigma_{\max}(t, s; w)$  is the optimum predecessor state for the hypothesis  $(t, s; w)$ . The back pointers  $B(t, s; w)$  are propagated simply according to the DP

decision. To include word boundaries, we introduce a special state  $s = 0$  which is used to start up a word. When encountering a potential word boundary, we have to perform the recombination over the predecessor words. To this purpose, we define:

$$H(w;t) := \max_v \{ p(w|v) \cdot Q(t, S(v); v) \} , \quad (11)$$

where  $p(w|v)$  are the conditional bigram probabilities as defined in Eq. (2). The symbol  $S(v)$  denotes the terminal state of word  $v$  from which new words can be started. To this purpose, we have to pass on both the score and the time index:

$$Q(t-1, 0; w) = H(w; t-1) \quad (12)$$

$$B(t-1, 0; w) = t-1 . \quad (13)$$

This equation assumes that first the normal states  $s = 1, \dots, S(w)$  are evaluated for each word  $w$  before the start-up states  $s = 0$  are evaluated. The same time index  $t$  is used intentionally, because the language model does not 'absorb' an acoustic vector. Note that the scores  $Q(t, s; w)$  capture both the acoustic observation dependent probabilities resulting from the HMM and the language model probabilities.

The operations to be performed are summarized in Table 1. The sequence of acoustic vectors extracted from the input speech signal is processed strictly from left to right. The search procedure works with a time-synchronous breadth-first strategy, i.e. all hypotheses for word sequences are extended in parallel for each incoming acoustic vector. To reduce the storage requirements, it is suitable to introduce back pointers and traceback arrays. The back pointers are propagated from state to state during the dynamic programming recursion and report the start time for each word end hypothesis. The traceback arrays are used to record the decisions about the best predecessor word for each word start-up. Using the back pointers and the traceback arrays, the recognized word sequence can be recovered efficiently by a few table look-ups into the traceback arrays at the end of the utterance.

Since all hypotheses cover the same portion of the input, their scores can be directly compared. This enables the system to avoid an exhaustive search and to perform a data driven search instead, i.e. to focus the search on those hypotheses which are most likely to result in the best state sequence [18]. Every 10-ms frame, the score of the best hypothesis is determined, then all hypotheses whose scores fall short of this optimal score by more than a fixed factor are pruned, i.e. are removed from further consideration. The experimental tests

Table 1 One-pass algorithm ('single best').

|                                          |                                    |                                                                      |
|------------------------------------------|------------------------------------|----------------------------------------------------------------------|
| proceed over time $t$ from left to right |                                    |                                                                      |
|                                          | ACOUSTIC LEVEL: process states     |                                                                      |
|                                          |                                    | - initialization: $Q(t-1, 0; w) = H(w; t-1)$<br>$B(t-1, 0; w) = t-1$ |
|                                          |                                    | - time alignment: $Q(t, s; w)$ using DP                              |
|                                          |                                    | - propagate back pointers $B(t, s; w)$                               |
|                                          |                                    | - prune unlikely hypotheses                                          |
|                                          |                                    | - purge bookkeeping lists                                            |
|                                          | WORD PAIR LEVEL: process word ends |                                                                      |
|                                          |                                    | for each pair $(w; t)$ do                                            |
|                                          |                                    | $H(w; t) = \max_v [p(w v) Q_v(t, S(w); w)]$                          |
|                                          |                                    | $v_0(w; t) = \arg \max_v [p(w v) Q_v(t, S(w); w)]$                   |
|                                          |                                    | - store best predecessor $v_0 := v_0(w; t)$                          |
|                                          |                                    | - store best boundary $\tau_0 := B(t, S(v_0); v_0)$                  |

indicate that for this type of beam search, depending on the acoustic input and the language model constraints, only a few percent of the potential state hypotheses have to be processed for every 10 ms of the input speech while at the same time the number of recognition errors is virtually not increased. To fully exploit the computational advantages of this beam search strategy, a list organization of the active search space [20] has been developed. At several levels such as words and states, lists are introduced to dynamically construct the search space such that the computational cost of the search is now proportional only to the number of active hypotheses and is independent of the overall size of the potential search space.

To simplify the presentation, we have not explicitly included intraphrase silence models. It is straightforward to extend the algorithm to allow for optional silence between words. This algorithm shown in Table 1 is still the most successful algorithm for both small-vocabulary and large-vocabulary speech recognition [15, 6, 14, 1, 7, 9, 11, 13, 30].

4 LEXICAL TREE AND ONE-PASS ALGORITHM

4.1 Lexical Pronunciation Tree

So far, we have considered a straightforward approach to organizing the search space, which was based on the use of a linear lexicon, i.e. each word is represented as a linear sequence of phonemes, independently of other words. In a 20,000-word vocabulary, there are typically many words that share the same beginning phonemes. Therefore a tree representation of the lexicon will be considered as an alternative to a linear lexicon. Such a lexical tree representation is an essential ingredient for handling vocabularies of 20,000 words and more (despite the apparent complications in the definition of the search space).

When analyzing the distribution of the state hypotheses over the state positions within a word, one typically finds in the experimental tests that the lion's share of the hypotheses covers the first and second phonemes in a word. In [10], for a 12,306-word dictation task, 79% and 16% of the state hypotheses were in the first and second phoneme, respectively. On the other hand, a large number of words in the 12,306-word vocabulary share the same initial sequence of phonemes. Therefore it seems promising to arrange the pronunciation lexicon as a tree. Each arc of the tree stands for a phoneme such that an arc sequence from the tree root to a tree leaf represents a legal phoneme sequence and thus a word of the vocabulary. For the 12,306-word vocabulary, Table 2 shows the distribution of arcs over the first 6 generation levels of the tree. The lexical tree consists of 43,000 arcs, which is a compression factor of 2.5 over the 108,000 phoneme copies of the linear lexicon. Even in the fourth level of the tree, the number (3,116) of phoneme arcs is only one quarter of the number (12,306) of words in the lexicon.

**Table 2** Distribution of the number of phoneme arcs for a 12,306-word lexicon using a tree organization.

|       |    |     |      |      |      |      |       |
|-------|----|-----|------|------|------|------|-------|
| Level | 1  | 2   | 3    | 4    | 5    | 6    | ≥ 7   |
| Arcs  | 28 | 331 | 1511 | 3116 | 4380 | 4950 | 29200 |

## 4.2 Predecessor Conditioned One-Pass Algorithm

For a unigram language model, we need only one tree, which is started up after incorporating the unigram language probability. However, for a bigram language model (and other more complicated language models), there is an added complication due to the fact that the identity of the word is known only at the end of a tree and only then can the language model probability be applied. Therefore the language model probabilities can only be incorporated after reaching the end, i.e. the final state, of the second word of the bigram. In other words, the use of the language model may be viewed as delayed by one word as compared to the bigram model without using a tree lexicon. To incorporate the bigram language model into the search, we will keep a separate copy of the lexical tree for each predecessor word. Fig. 6 illustrates this concept for a three-word vocabulary ( $A, B, C$ ). For each predecessor word  $v$ , there is a separate lexical tree so that during the search process we always know the predecessor word  $v$  when a word end hypothesis  $w$  is hypothesized. In the set-up of Fig. 6, we apply the bigram probability  $p(w|v)$  when the final state of word  $w$  with predecessor  $v$  has been reached, and use the resulting overall score to start up the corresponding lexical tree, i.e. the tree that has word  $w$  as predecessor.

The key quantities for dynamic programming are again the partial scores  $Q(t, s; w)$  and the back pointers  $B(t, s; w)$ . However, the quantities must be made dependent on the predecessor word  $v$  to account for the delayed application of the bigram language model:

$Q_v(t, s; w) :=$  score of the best path up to time  $t$  that ends in state  $s$  of word  $w$  for predecessor  $v$ .

$B_v(t, s; w) :=$  starting time of the best path up to time  $t$  that ends in state  $s$  of word  $w$  for predecessor  $v$ .

Both quantities are evaluated using the dynamic programming recursion for  $Q_v(t, s; w)$ :

$$Q_v(t, s; w) = \max_{\sigma} \{ q(x_t, s | \sigma; w) \cdot Q_v(t-1, \sigma; w) \} \quad (14)$$

$$B_v(t, s; w) = B_v(t-1, \sigma_{\max}(t, s; w, v); w), \quad (15)$$

where  $\sigma_{\max}(t, s; w, v)$  is the optimum predecessor state for the hypothesis  $(t, s; w)$  and predecessor word  $v$ . The back pointers  $B_v(t, s; w)$  are propagated

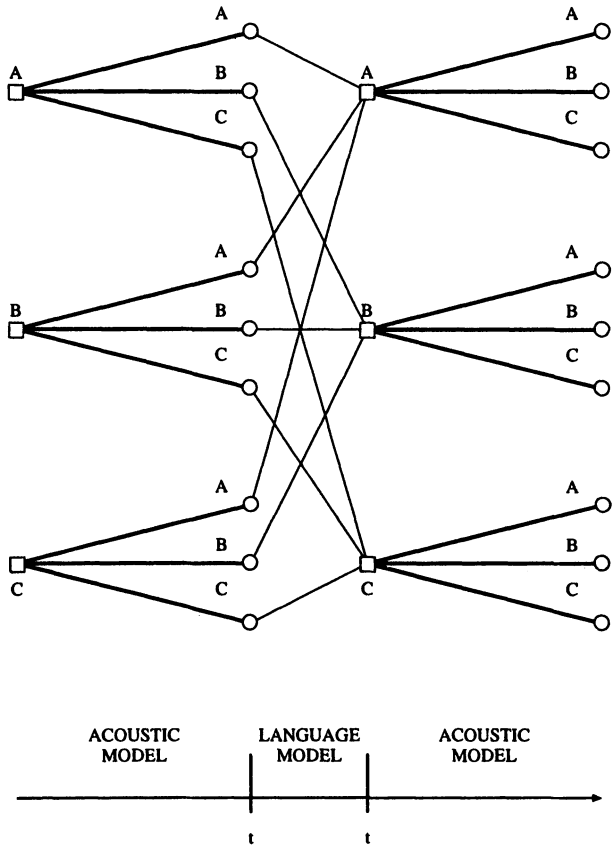


**Figure 6** Word recombination for a lexical tree

according to the DP decision. Unlike the predecessor word  $v$ , the index  $w$  for the word under consideration is not really needed, because from the phoneme arcs of each tree, the corresponding words can be uniquely recovered. However, for notational consistency, we have retained both indices  $v$  and  $w$ .

Using a suitable initialization for  $\sigma = 0$ , this equation includes the optimization over the unknown word boundaries. At word boundaries, we have to find the best predecessor word for each word. To this purpose, we define:

$$H(w;t) := \max_v \{ p(w|v) \cdot Q_v(t, S(w); w) \} . \tag{16}$$

To start up words, we have to pass on the score and the time index:

$$Q_v(t-1, 0; w) = H(v; t-1) \quad (17)$$

$$B_v(t-1, 0; w) = t-1. \quad (18)$$

Note that here the right-hand side does not depend on word index  $w$  due to the notational scheme explained above.

### 4.3 Experimental Results

Systematic experimental tests have been carried out for the search methods presented so far [22, 20]; here, we only summarize the most important results:

- Throughout all recognition tests, ranging from 100 to 10,000 vocabulary words, the experimental tests indicate that by using the beam search strategy, only a small fraction, typically a few percent or less, of the potential state hypotheses have to be processed while at the same time the number of recognition errors is virtually not increased over a full search.
- For a 12,000-word dictation task, a series of experiments was run to study the effect of the tree lexicon on the search effort [20]. By using the tree lexicon, the search effort was reduced by a factor of 6 over the linear lexicon. A further reduction by a factor of 3 was obtained by applying a look-ahead technique at the phoneme level in the lexical tree [10].
- The dramatic reduction for a tree lexicon must be attributed to the subtle interaction between the focusing property of beam search and the organization of the search space. The ambiguity in the decisions and therefore the number of hypotheses is particularly high at the beginning of words; by a tree lexicon, such hypotheses are compressed in a most efficient way. In other words, the tree lexicon leads to a search space that is most compact where the highest uncertainty is expected, i.e. at the beginning portions of words.
- In general, when more detailed models with a potentially larger search space are used, the effort of the beam search strategy is not much affected. For example, for a linear lexicon, when going from a bigram to a trigram model, the search effort remained the same for the particular experiments [22]. For a tree lexicon, when switching from a unigram tree to the bigram tree copies, the search effort was increased by a factor of 2 only [20]. In other words, the early use of more detailed models results in a more focussed beam search despite the increase in the potential search space.

- As to the pruning strategy, we have found that it is advantageous to use two pruning thresholds: an acoustic pruning threshold for the intraword recombinations and a language pruning threshold for the interword recombinations [28]. This second threshold is related to an anticipated language model probability, since a state hypothesis at any point in the lexical tree implies a certain lower bound for the language model probability. Ultimately, the choice of the pruning thresholds is based on trial and error, although the choice is not critical [28].
- There is a (close to) real time implementation of the search strategy described in this chapter; the hardware is based on an Intel 80486 microprocessor and 16 Mbyte RAM for a 25,000-word continuous speech recognition system [27].

## 5 WORD GRAPHS AND ONE-PASS ALGORITHM

### 5.1 Specification of Word Graphs

The main idea of a word graph is to come up with word alternatives in regions of the speech signal, where the ambiguity in the acoustic recognition is high. The advantage is that the acoustic recognition is decoupled from the application of the language model and that a complex language model, in particular a long-span language model, can be applied in a subsequent postprocessing step. The number of word alternatives should be adapted to the level of ambiguity in the acoustic recognition. The difficulty in efficiently constructing a good word graph is the following: the start time of a word depends in general on the predecessor words. In a first approximation, we limit this dependence to the immediate predecessor word and obtain the so-called word pair approximation:

Given a word pair and its ending time, the word boundary between the two words is independent of the further predecessor words.

This word pair approximation was originally introduced in [24] to efficiently calculate multiple or  $n$ -best sentences. The word graph can be expected to be more efficient than the  $n$ -best approach. In the word graph approach, word hypotheses need to be generated only locally whereas in  $n$ -best methods each local alternative requires a whole sentence to be added to the  $n$ -best list. To

give a (over) simplified example, suppose we have 10 spoken words and 2 word hypotheses for each word position. The  $n$ -best method then requires  $2^{10}=1024$  sentence hypotheses, whereas the word graph approach produces a graph of only  $2 \cdot 10 = 20$  word arcs.

There have been a number of attempts at using two explicit levels in search: the first level producing a short list of either single word or sentence hypotheses, and a second level where the final decision is taken using a complex language model [23, 24, 25, 8, 21, 17].

As in [21], we prefer the term ‘word graph’ to ‘word lattice’ to indicate that when combining word hypotheses, we do not allow overlaps or gaps along the time axis. The difference to the word graph algorithm in [21] is that we use a formal specification of word graphs and make explicit use of the word pair approximation. As a result, we can incorporate the word graph construction into our standard one-pass algorithm using predecessor conditioned lexical trees, whereas in [21], the search was based on time conditioned lexical trees.

In this section, we will formally specify the word graph construction problem and pave the way for the word graph algorithm. We start with the fundamental problem of word graph construction:

- Given a word  $w$  and ending time  $t$ , how can we find a limited number of ‘most likely’ predecessor words?
- This task is difficult since the start time of word  $w$  may very well depend on the predecessor word under consideration, which results in an interdependence of start times and predecessor words.

In view of the most successful one-pass beam search strategy, what we want to achieve conceptually, is to keep track of word sequence hypotheses whose scores are very close to the locally optimal hypothesis, but that do not survive due to the recombination process.

The basic idea is to represent all these word sequences by a word graph, in which each arc represents a word hypothesis. Each word sequence contained in the word graph should be close (in terms of scoring) to the single best sentence produced by the one-pass algorithm. To describe the word graph construction algorithm, we introduce the following definitions:

- $h(w; \tau, t) := Pr(x_{\tau+1}^t | w)$  = probability that word  $w$  produces the acoustic vectors  $x_{\tau+1} \dots x_t$ .
- $G(w_1^n; t) := Pr(w_1^n) \cdot Pr(x_1^t | w_1^n)$  = (joint) probability of generating the acoustic vectors  $x_1 \dots x_t$  and a word sequence  $w_1^n$  with ending time  $t$ .

Using these definitions, we can isolate the probability contributions of a particular word hypothesis with respect to both the language model and the acoustic model. This decomposition can be visualized as follows:

$$\underbrace{x_1, \dots, x_\tau}_{G(w_1^{n-1}; \tau)} \underbrace{x_{\tau+1}, \dots, x_t}_{h(w_n; \tau, t)} \underbrace{x_{t+1}, \dots, x_T}_{\dots} \quad (19)$$

Exploiting an  $m$ -gram language model  $p(u_m | u_1^{m-1})$ , we can recombine word sequence hypotheses at the phrase level if they do not differ in their final  $(m-1)$  words. Therefore it is sufficient to distinguish partial word sequence hypotheses only by their final words  $u_2^m$ . The corresponding score is denoted by  $H(u_2^m; t)$  and is defined as the joint probability of generating the acoustic vectors  $x_1 \dots x_t$  and a word sequence with *ending* sequence  $u_2^m$  and *ending* time  $t$ :

$$H(u_2^m; t) := \max_{w_1^n} \{ Pr(w_1^n) \cdot Pr(x_1^t | w_1^n) : w_{n-m+2}^n = u_2^m \} , \quad (20)$$

where, as expressed by the notation, the final portion  $u_2^m$  of the word sequence  $w_1^n$  is not subjected to the maximization operation. Note that the quantity  $H(u_2^m; t)$  is similar to the definition introduced for the single-best one-pass algorithm in Section 3. Using the above definition, we can write the dynamic programming equation at the word level:

$$H(u_2^m; t) = \max_{u_1} \{ p(u_m | u_1^{m-1}) H(u_1^{m-1}; \tau(t; u_1^m)) h(u_m; \tau(t; u_1^m), t) \} . \quad (21)$$

Here we have used the function  $\tau(t; u_1^m)$  to denote the word boundary between  $u_{m-1}$  and  $u_m$  for the word sequence with final portion  $u_1^m$  and ending time  $t$ . Note that we have included the language model to achieve a better pruning effect. The word boundary itself is defined by a maximization operation:

$$\tau(t; u_1^m) := \arg \max_{\tau} \{ H(u_1^{m-1}; \tau) h(u_m; \tau, t) \} . \quad (22)$$

So far this has been just a notational scheme for the word boundary function  $\tau(t; u_1^m)$ . The crucial assumption now is that the dependence of the word

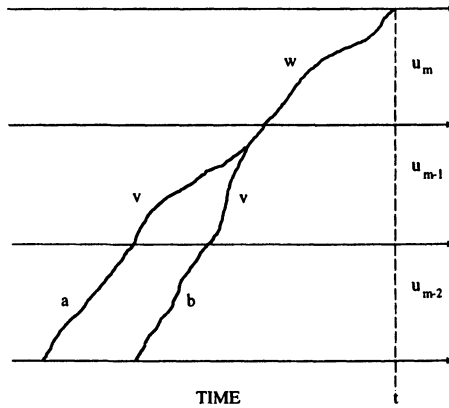


**Figure 7** Word pair approximation.

boundary  $\tau(t; u_1^m)$  can be confined to the final word pair  $u_{m-1}^m$ . The justification is that the other words have virtually no effect on the position of the word boundary in the word pair  $(u_{m-1}, u_m)$  [24]. This so-called word pair approximation is illustrated in Fig. 7. From this figure, it is obvious that the assumption of the word pair approximation is satisfied if the predecessor word  $u_{m-1}$  is sufficiently long: all time alignment paths then are recombined before they reach the final state of the predecessor word. We express this word pair approximation by the equation:

$$\tau(t; u_1^m) = \text{const}(u_1^{m-2}) . \quad (23)$$

As long as only a bigram language model is used, the word pair approximation is still exact (assuming a conservatively large pruning threshold). An even further simplification is the single word approximation used in [26].

## 5.2 Word Graph Algorithm

The word boundary function  $\tau(t; v, w)$  has not yet been specified. In principle, it can be computed using either the so-called two-level algorithm [23] or the one-pass algorithm describe before, which both compute only the best single word sequence. However, to apply beam search, it is more convenient to use the one-pass algorithm. Since the hypotheses must be distinguished by the predecessor word, we have to use the predecessor conditioned variant of the one-pass algorithm as presented in the preceding section.

To extend the predecessor conditioned algorithm into an algorithm for word graph construction, we have only to add the two equations for calculating the word boundary function  $\tau(t; v, w)$  and the word score  $h(w; \tau, t)$ . The word boundaries are obtained using the back pointers at the word ends:

$$\tau(t; v, w) = B_v(t, S(w); w) . \quad (24)$$

For each predecessor word  $v$  along with word boundary  $\tau = \tau(t; v, w)$ , the word scores are recovered using the equation:

$$h(w; \tau, t) := \frac{Q_v(t, S(w); w)}{H(v; \tau)} , \quad (25)$$

where we obtain  $H(w; t)$  as usual:

$$H(w; t) = \max_v \{ p(w|v) \cdot Q_v(t, S(w); w) \} . \quad (26)$$

The details of the algorithm are summarized in Table 3. The operations are organized in two levels: the acoustic level and the word pair level. At the end of the utterance, the word graph is constructed by tracing back through the bookkeeping lists. A third level, the phrase level, has been included as option for the final recognition. There is only a gradual difference between the following two strategies:

- *Extended one-pass approach:* The word pair approximation serves only as a simplification in the one-pass strategy to avoid too many copies when multiplying out the constraints.
- *Two-pass approach:* First, a word graph is constructed. Then, at the so-called phrase level, the best sentence is computed using a more complex language model.

What has to be added to the single-best one-pass strategy, is the bookkeeping at the word level: rather than just the best surviving hypothesis, the algorithm has to memorize all the word sequence hypotheses that are recombined into just one hypothesis to start up the next lexical tree (or word models). In the single-best method, only the surviving hypothesis  $[v_0, \tau_0]$  has to be kept track of. To reduce the memory cost, a method for removing obsolete hypotheses must be included along with list organizations and dynamic construction of the search space [20]. Given the word graph and the language model, the final search can be carried out at the word level using a left-to-right dynamic programming

**Table 3** One-pass algorithm for word graph construction ('single best' and 'word graph').

| proceed over time $t$ from left to right |                                    |                                                                                                                    |
|------------------------------------------|------------------------------------|--------------------------------------------------------------------------------------------------------------------|
|                                          | ACOUSTIC LEVEL: process states     |                                                                                                                    |
|                                          |                                    | - initialization: $Q_v(t-1, 0; w) = H(v; t-1)$<br>$B_v(t-1, 0; w) = t-1$                                           |
|                                          |                                    | - time alignment: $Q_v(t, s; w)$ using DP                                                                          |
|                                          |                                    | - propagate back pointers $B_v(t, s; w)$                                                                           |
|                                          |                                    | - prune unlikely hypotheses                                                                                        |
|                                          |                                    | - purge bookkeeping lists                                                                                          |
|                                          | WORD PAIR LEVEL: process word ends |                                                                                                                    |
|                                          |                                    | single best: for each pair $(w; t)$ do                                                                             |
|                                          |                                    | $H(w; t) = \max_v [p(w v) Q_v(t, S(w); w)]$<br>$v_0(w; t) = \arg \max_v [p(w v) Q_v(t, S(w); w)]$                  |
|                                          |                                    | - store best predecessor $v_0 := v_0(w; t)$                                                                        |
|                                          |                                    | - store best boundary $\tau_0 := B_{v_0}(t, S(w); w)$                                                              |
|                                          |                                    | word graph: for each triple $(t; v, w)$ store                                                                      |
|                                          |                                    | - word boundary $\tau(t; v, w) := B_v(t, S(w); w)$<br>- word score $h(w; \tau, t) := Q_v(t, S(w); w) / H(v; \tau)$ |
|                                          | PHRASE LEVEL search (optional)     |                                                                                                                    |

algorithm. The cost of this search through the word graph depends on the complexity of the language model. For a trigram model, it is typically less than 1% of the effort for constructing the word graph.

An example of a word graph for a three-word vocabulary  $A, B, C$  (including silence at the sentence beginning and end) is shown in Fig. 8. The arcs stand for word hypotheses, where the circles along with the word name denote the word end. Note the following properties, which are a result of the word graph algorithm:

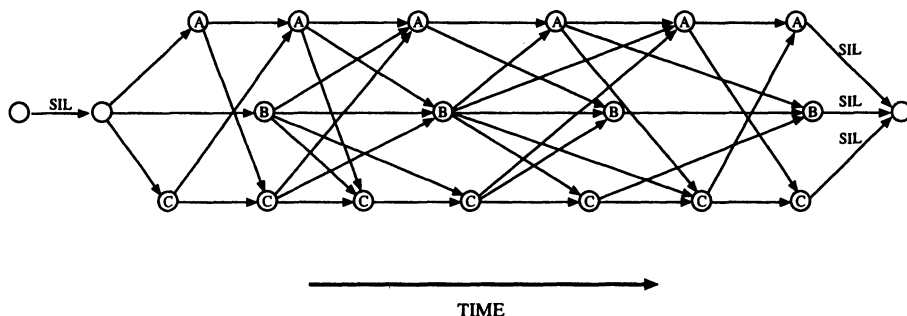


**Figure 8** Example of a word graph.

- There is a maximum for the number of incoming word arcs, namely the vocabulary size which is the maximum number of possible predecessor words.
- There is no maximum for the number of outgoing word arcs; this effect is due to the fact that, even for the same predecessor word, a word can have different ending time hypotheses.
- For clarity, we have not included intraphrase silence models in the presentation of the algorithm. Silence models, which must also be made dependent on the predecessor words [10], are treated like normal words of the vocabulary.

There are two more refinements which suggest themselves:

- For short or function words like articles and prepositions, the quality of the word pair approximation might be questionable, and word triples or higher  $m$ -grams might be used instead in these cases.
- Long words with identical ending portion may waste search effort and could be merged when forming word pairs in the word graph algorithm.

In both cases, the obvious remedy is to make the word copies dependent on a suitably defined history using the phonetic script of the predecessor words.

**Table 4** Recognition results for the word graph method on the Nov. 92 WSJ task.

| 5k Closed Vocabulary   |      |      | 20k Open Vocabulary    |      |      |
|------------------------|------|------|------------------------|------|------|
| 330 sent. = 5353 words |      |      | 333 sent. = 5643 words |      |      |
| BI = 111; TRI = 57     |      |      | BI = 205; TRI = 140    |      |      |
| WGD                    | G[%] | W[%] | WGD                    | G[%] | W[%] |
| 1462.0                 | 3.3  | 4.9  | 2112.0                 | 5.3  | 11.9 |
| 130.0                  | 3.6  | 4.9  | 875.0                  | 5.7  | 11.9 |
| 35.0                   | 5.8  | 4.9  | 483.0                  | 5.7  | 11.9 |
| 14.0                   | 7.9  | 4.9  | 39.0                   | 10.6 | 11.9 |
| 6.1                    | 8.5  | 5.0  | 14.5                   | 20.0 | 11.9 |
| 3.0                    | 22.0 | 5.1  | 5.9                    | 25.0 | 11.8 |
| 1.9                    | 33.0 | 5.7  | 3.9                    | 30.0 | 11.9 |
| 1.5                    | 45.0 | 6.4  | 2.8                    | 37.0 | 12.0 |
| 1.3                    | 54.0 | 7.0  | 1.7                    | 51.0 | 12.9 |
| 1.2                    | 56.0 | 7.3  | 1.4                    | 59.0 | 13.5 |
| Bigram                 |      | 7.3  | Bigram                 |      | 14.1 |

### 5.3 Experimental Results

The word graph algorithm has been tested on the WSJ task using the November '92 evaluation set (4 male and 4 female speakers). The details of the acoustic modeling and training are described in [1]. The word graph was constructed using a bigram model. To study the effect of the word graph size on the word error rate, a conservatively large word graph was stored on disk for each test sentence. Then by varying the language model pruning threshold [28], the graph was reduced and the effect on the final word recognition error rate was measured after applying a trigram model.

Table 4 summarizes these results: graph sentence error rate (G[%]), i.e. the fraction of cases where the spoken sentence is not contained in the word graph, and recognition word error rate (W[%]) are shown as a function of the word graph density (WGD) which is defined as the total number of word graph arcs divided by the number of spoken words. For the open vocabulary test, only the sentences without any unknown word were used. In the word graph density, the silence models are included. The perplexities of the bigram and trigram language models used are denoted as BI and TRI in Table 4. As can be seen from the table, the word graph density can be reduced down to 14 for the

5,000-word (5k) closed vocabulary test and to 4 for the 20,000-word (20k) open vocabulary test without affecting recognition accuracy.

Instead of the sentence errors, we can also measure the word errors for a given word graph. This is achieved by determining that sentence through the word graph that best matches the spoken sentence where the matching criterion is defined in terms of word substitutions, deletions and insertions. This measure provides an estimate of the minimum word error rate for this word graph. For the best experiment reported in Table 4, the word graph error rate is about 1% for the 5k closed lexicon and about 4% for the 20k open lexicon (including the unknown words).

## 6 CONCLUSIONS

Conceptually, the dynamic programming search strategies described in this chapter amount to the (implicit or dynamic) construction of a huge graph of expected acoustic segments to achieve what could be called constraint satisfaction for all available knowledge sources. Although these concepts or similar ones had already been published in 1971 and used successfully for small-vocabulary speech recognition around 1980 by several researchers, they were not generally expected to be appropriate for 20,000-word, natural language speech recognition. In light of this history, it is interesting to remember what Dennis Klatt said about the underlying principles of the DRAGON and HARP systems developed in 1976 [12, p. 261/2]:

“...the application of simple structured models to speech recognition. It might seem to someone versed in the intricacies of phonology and the acoustic-phonetic characteristics of speech that a search of a graph of expected acoustic segments is a naive and foolish technique to use to decode a sentence. In fact such a graph and search strategy (and probably a number of other simple models) can be constructed and made to work very well indeed if the proper acoustic-phonetic details are embodied in the structure.”

## REFERENCES

- [1] X. Aubert, C. Dugast, H. Ney, V. Steinbiss: Large Vocabulary, Continuous Speech Recognition of Wall Street Journal Corpus. Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing, Adelaide, Australia, Vol. II, pp. 129-132, April 1994.
- [2] L. R. Bahl, F. Jelinek, R. L. Mercer: A Maximum Likelihood Approach to Continuous Speech Recognition. IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 5, pp. 179-190, March 1983.
- [3] J. K. Baker: Stochastic Modeling for Automatic Speech Understanding, in D. R. Reddy (ed.): 'Speech Recognition', Academic Press, New York, pp. 512-542, 1975.
- [4] R. E. Bellman: 'Dynamic Programming', Princeton University Press, Princeton, NJ, 1957.
- [5] J. S. Bridle, M. D. Brown, R. M. Chamberlain: An Algorithm for Connected Word Recognition. Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing, Paris, pp. 899-902, May 1982.
- [6] R. Cardin, Y. Normandin, R. DeMori: High Performance Connected Digit Recognition using Codebook Exponents. Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing, San Francisco, CA, Vol. I, pp. 505-508, March 1992.
- [7] V. Digalakis, H. Murveit: Genones: Optimizing the Degree of Mixture Tying in a Large Vocabulary Hidden Markov Model-Based Speech Recognizer. Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing, Adelaide, Australia, Vol. I, pp. 537-540, April 1994.
- [8] L. Fissore, E. Giachin, P. Laface, P. Massafra: Using Grammars in Forward and Backward Search. Proc. Europ. Conf. on Speech Communication and Technology, Berlin, pp. 1525-1528, Sep. 1993.
- [9] J. L. Gauvain, L. F. Lamel, G. Adda, M. Adda-Decker: The LIMSI Speech Dictation System: Evaluation on the ARPA Wall Street Journal Task. Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing, Adelaide, Australia, Vol. I, pp. 557-560, April 1994.
- [10] R. Haeb-Umbach, H. Ney: Improvements in Time-Synchronous Beam Search for 10,000-Word Continuous Speech Recognition. IEEE Trans. on Speech and Audio Processing, Vol. 2, pp. 353-356, April 1994.

- [11] M.-Y Hwang, R. Rosenfeld, E. Thayer, R. Mosur, L. Chase, R. Weide, X. Huang, F. Alleva: Improving Speech Recognition Performance via Phone-Dependent VQ Codebooks and Adaptive Language Models in SPHINX II. Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing, Adelaide, Australia, Vol. I, pp. 549-552, April 1994.
- [12] D. Klatt: Overview of the ARPA Speech Understanding Project, pp. 249-271, in W. A. Lea: 'Trends in Speech Recognition', Prentice-Hall, Englewood Cliffs, NJ, 1980.
- [13] F. Kubala, A. Anastasakos, J. Makhoul, L. Nguyen, R. Schwartz: Comparative Experiments on Large Vocabulary Speech Recognition. Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing, Adelaide, Australia, Vol. I, pp. 561-564, April 1994.
- [14] C.-H. Lee, E. Giachin, L. R. Rabiner, R. Pieracini, A. E. Rosenberg: Improved Acoustic Modeling for Large Vocabulary Continuous Speech Recognition. Computer Speech and Language, Vol. 6, No. 2, pp. 103-127, April 1992.
- [15] C. H. Lee, L. R. Rabiner: A Frame-Synchronous Network Search Algorithm for Continuous Word Recognition. IEEE Trans. on Acoustics, Speech and Signal Processing, Vol. ASSP-37, No. 11, pp. 1649-1658, Nov. 1989.
- [16] S. E. Levinson, L. R. Rabiner, M. M. Sondhi: An Introduction to the Application of the Theory of Probabilistic Functions of a Markov Process to Automatic Speech Recognition. The Bell System Technical Journal, Vol. 62, No. 4, pp. 1035- 1074, April 1983.
- [17] A. Ljolje, M. Riley, D. Hindle, F. Pereira: The AT&T 60,000 Word Speech-To-Text System. Proc. ARPA Spoken Language Systems Technology Workshop, Austin, TX, pp. 162-165, Jan. 1995.
- [18] B. T. Lowerre, R. Reddy: The HARPY Speech Understanding System, pp. 340-360, in W. A. Lea (ed.): 'Trends in Speech Recognition', Prentice-Hall, Englewood Cliffs, NJ, 1980.
- [19] H. Ney: The Use of a One-Stage Dynamic Programming Algorithm for Connected Word Recognition. IEEE Trans. on Acoustics, Speech and Signal Processing, Vol. ASSP-32, No. 2, pp. 263-271, April 1984.
- [20] H. Ney, D. Mergel, A. Noll, A. Paeseler: Data Driven Organization of the Dynamic Programming Beam Search for Continuous Speech Recognition. IEEE Trans. on Signal Processing, Vol. SP-40, No. 2, pp. 272-281, Feb. 1992.

- [21] M. Oerder, H. Ney: Word Graphs: An Efficient Interface Between Continuous Speech Recognition and Language Understanding. Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing, Minneapolis, MN, Vol. II, pp. 119-122, April 1993.
- [22] A. Paeseler, H. Ney: Continuous Speech Recognition Using a Stochastic Language Model. Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing, Glasgow, pp. 719-722, April 1989.
- [23] H. Sakoe: Two-Level DP Matching — A Dynamic Programming-Based Pattern Matching Algorithm for Connected Word Recognition. IEEE Trans. on Acoustics, Speech and Signal Processing, Vol. ASSP-27, pp. 588-595, Dec. 1979.
- [24] R. Schwartz, S. Austin: A Comparison of Several Approximate Algorithms for Finding Multiple (N-Best) Sentence Hypotheses. Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing, Toronto, pp. 701-704, May 1991.
- [25] F. K. Soong, E.-F. Huang: A Tree-Trellis Fast Search for Finding the  $N$ -Best Sentence Hypotheses. Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing, Toronto, pp. 705-708, May 1991.
- [26] V. Steinbiss: A Search Organization for Large Vocabulary Recognition Based upon  $N$ -Best Decoding. Proc. of the 2nd European Conf. on Speech Communication and Technology, Genova, Vol. 3, pp. 1217-1220, Sep. 1991.
- [27] V. Steinbiss, H. Ney, R. Haeb-Umbach, B.-H. Tran, U. Essen, R. Kneser et al.: The Philips Research System for Large-Vocabulary Continuous-Speech Recognition. Third European Conference on Speech Communication and Technology, Berlin, pp. 2125-2128, Sep. 1993.
- [28] V. Steinbiss, B.-H. Tran, H. Ney: Improvements in Beam Search. Proc. Int. Conf. on Spoken Language Processing, Yokohama, Japan, pp. 1355-1358, Sep. 1994.
- [29] T. K. Vintsyuk: Elementwise Recognition of Continuous Speech Composed of Words from a Specified Dictionary. Cybernetics, Vol. 7, pp. 133-143, March-April 1971.
- [30] Ph. Woodland, J. J. Odell, V. Valtech, S. J. Young: Large Vocabulary Continuous Speech Recognition Using HTK. Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing, Adelaide, Australia, Vol. II, pp. 125-128, April 1994.