Computer Speech and Language (1997) **11,** 43–72

A word graph algorithm for large vocabulary continuous speech recognition

Stefan Ortmanns, Hermann Ney

Lehrstuhl für Informatik VI, RWTH Aachen, University of Technology, D-52056 Aachen, Germany

Xavier Aubert

Philips GmbH Forschungslaboratorien, Weißhausstraße 2, D-52066 Aachen, Germany

Abstract

This paper describes a method for the construction of a word graph (or lattice) for large vocabulary, continuous speech recognition. The advantage of a word graph is that a fairly good degree of decoupling between acoustic recognition at the 10-ms level and the final search at the word level using a complicated language model can be achieved. The word graph algorithm is obtained as an extension of the one-pass beam search strategy using word dependent copies of the word models or lexical trees.

The method has been tested successfully on the 20 000-word NAB'94 task (American English, continuous speech, 20 000 words, speaker independent) and compared with the integrated method. The experiments show that the word graph density can be reduced to an average number of about 10 word hypotheses, i.e. word edges in the graph, per spoken word with virtually no loss in recognition performance.

1. Introduction

Dynamic programming was already used in the early days of automatic speech recognition (Vintsyuk, 1971; Baker, 1975; Sakoe, 1979; Bridle *et al.*, 1982; Ney, 1982) for highly constrained small vocabulary tasks. In the last decade, it was successfully extended to handle high-perplexity natural language speech recognition. Meanwhile, 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. Despite these successes, there was always the objection that dynamic programming produces only the single most likely sentence and nothing else. In particular, in many recognition tasks, it might be desirable to produce a list of the n best ranking sentence

0885–2308/97/010043 + 30 $25.00/0/1a960022

alternatives or a word graph each edge of which stands for a hypothesis about a spoken word.

In this paper, we show how the concept of dynamic programming for determining the single best sentence can be extended in a natural way to produce a word graph. 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 had originally been introduced by Schwartz & Austin (1991) 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 an (over) simplified example, suppose we have 10 spoken words and two 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 edges.

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 (Sakoe, 1979; Schwartz & Austin, 1991; Soong & Huang, 1991; Fissore *et al.*, 1993; Oerder & Ney, 1993; Ljolje *et al.*, 1995). The novel contributions of this paper are the following.

- We give a consistent and formal specification of a word graph. In particular, the word graph will be defined using a word boundary definition.
- Using this formal specification of a word graph, there is a natural way to control the trade-off between the accuracy and coverage of the word graph and the cost of its generation. We argue that in most cases the so-called word pair approximation is a suitable assumption, as will be confirmed by the experiments, which results in a very efficient algorithm for word graph construction.
- We show that this word graph concept can be embedded in the successful one-pass beam search strategy, in particular in the context of a word conditioned search strategy that makes use of a prefix tree organization of the pronunciation lexicon. It will be shown that basically only the bookkeeping technique has to be modified to employ the word conditioned one-pass algorithm for word graph construction.
- We present experimental results for the 20 000-word North American Business (NAB'94) task, which demonstrate the high performance and high efficiency of the word graph method proposed.

TABLE I. Definitions and explanations of the most common terms

One-pass vs. multi-pass: we call a search a one-pass strategy if there is one single pass over the input sentence as opposed to a multi-pass or multi-level concept. The one-pass search strategy is virtually always based on dynamic programming.
Time-synchronous: a search strategy is called time-synchronous if the search hypotheses are formed in a time-synchronous fashion over the sequence of acoustic vectors. Typically, the time-synchronous concept goes hand in hand with the one-pass search strategy. An example of a search strategy that is *not* necessarily time-synchronous is given by the A^* search or stack decoding.
Integrated search: we call a search strategy integrated if *all* available knowledge sources, e.g. acoustic–phonetic models, the constraints of the pronunciation lexicon and the language model, are exploited in the search process at the same time; typically this concept is implemented in a one-pass strategy.
Word-conditioned vs. time-conditioned: these terms refer to the way in which the search space, especially in the context of dynamic programming, is structured. In a word-conditioned search, each search hypothesis is conditioned on the predecessor word. This implies that the optimization over the unknown ending time of the predecessor word, i.e. the word boundary between the predecessor word and the word under consideration, has already been carried out in an early phase of the search procedure. This method is therefore different from a time-conditioned search, where for each search hypothesis the dependence on the ending time of the predecessor word is explicitly retained and the optimization over the unknown word boundaries is performed as a final step of the search.
Single best: by single best, we mean a search concept which determines the single most likely word sequence. The alternatives are, among others, n-best concepts and word graph methods.
Word graph: the idea here is to organize the high-ranking sentence hypotheses in the form of a graph whose edges represent the hypothesized single words. Sometimes, the term "word lattice" is used synonymously. However, in this paper, by the term "word graph" we imply that gaps or overlaps between word hypotheses are not allowed.

As in Oerder and Ney (1993), 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 Oerder and Ney (1993) 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 the standard one-pass algorithm using word conditioned lexical trees, whereas in Oerder and Ney (1993) the search was based on time-conditioned lexical trees.

A few words about the terminology in the context of the various search concepts seem appropriate. These special terms do not necessarily have strict mathematical definitions, but are primarily used to highlight certain aspects of the search strategy considered. As a result, these terms may have some overlap in their meanings. We try to give short explanations of the most common terms in Table I.

The organization of the paper is as follows. To lay the ground for the word graph method, in Section 2, we review the one-pass beam search using a tree organization of the pronunciation lexicon and extend it from bigram language models to trigram language models. In Section 3, we present the basic concept for the specification of word graphs. To this purpose, we introduce the so-called word boundary function. Using the word pair approximation, we show how this approach can be incorporated into the one-pass beam search in a natural way, which is to change the bookkeeping scheme at word boundary hypotheses. In Section 4, we present experimental results on the NAB'94 20 000-word development using the word graph method designed in Section

3. In addition to a trigram language model, we use the word graph method in connection with a unigram/bigram cache language model. For the case of a trigram language model, we give a detailed comparison of the word graph method and the integrated method.

2. Review of the integrated method: one-pass search for the single-best sentence

As will be shown later, the algorithm for constructing a word graph turns out to be an extension of single-best dynamic programming search. Therefore, in this section, we review this one-pass single-best integrated search strategy, which as usual is combined with a beam search concept for removing unlikely search hypotheses. This will be done first for a bigram language model, and then we will show how to integrate a trigram language model into such a search concept. Both variants of the integrated search will be needed later: the bigram version paves the way for the word graph algorithm, and the trigram version of the integrated search will be used in the experiments to compare the word graph method and the integrated method.

This one-pass dynamic programming algorithm provides the basic component of the search component in many successful systems for both small-vocabulary and large-vocabulary speech recognition (Cardin *et al.*, 1992; Lee *et al.*, 1992; Ney *et al.*, 1992*a*; Alleva *et al.*, 1993; Murveit *et al.*, 1993; Aubert *et al.*, 1994; Gauvain *et al.*, 1994; Kubala *et al.*, 1994; Woodland *et al.*, 1994; Ortmanns & Ney, 1995).

When applying this algorithm to large-vocabulary recognition, say a 20 000 word-task, it seems natural and very desirable for efficiency reasons to organize the pronunciation lexicon in the form of a prefix tree, in which each arc represents a phoneme model, be it context dependent or independent (Ney *et al.*, 1992*a*; Haeb-Umbach & Ney, 1994; Ortmanns & Ney, 1995). This idea of using a tree representation was already suggested in the seventies in the CASPERS system (Klovstad & Mondshein, 1975) and in the LAFS (lexical access from spectra) system (Klatt, 1980). However, when using such a lexical tree in the framework of a language model, e.g. a bigram model, and dynamic programming, there are technical details that have to be taken into account and require a careful structuring of the search space (Ney *et al.*, 1992*a*; Haeb-Umbach & Ney, 1994). Next we will review the full details of the search algorithm for such a context.

2.1. Word conditioned lexical tree search algorithm

When using a bigram language model in connection with such a tree representation of the pronunciation lexicon, we face the problem that the identity of the hypothesized word w is known only when a leaf of the tree has been reached. Therefore the language model probabilities can only be fully incorporated after reaching the terminal state of the second word of the bigram. As a result, we can apply the language model probability only at the end of a tree. To make the application of the dynamic programming principles possible, we structure the search space as follows. For each predecessor word v, we introduce a separate copy of the lexical tree so that during the search process we always know the predecessor word v when a word end hypothesis w is hypothesized.

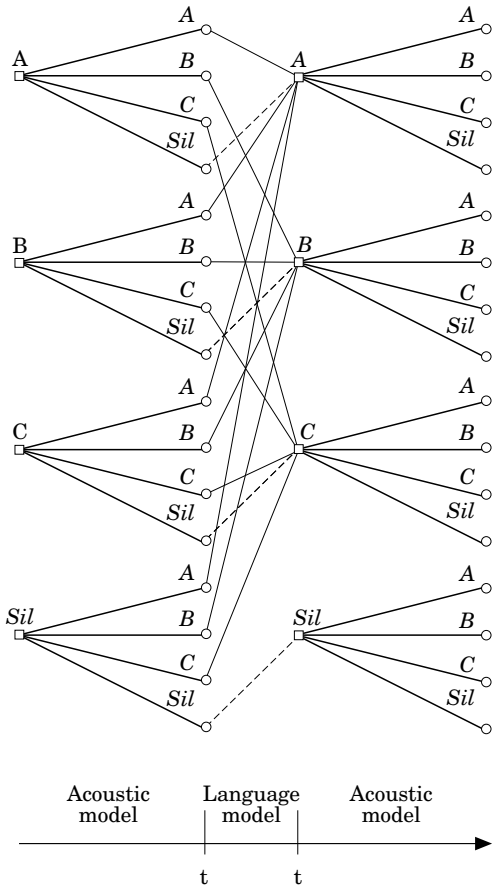

Figure 1. Bigram language model recombination and intraphrase silence (*Sil*) handling for a tree lexicon (three-word vocabulary: *A*, *B*, *C*) using word conditioned tree copies. ——: Acoustic recombination within a tree copy; ——: bigram language model recombination; – – –: word boundary recombination for an interphase silence copy; □: root of a tree copy; ○: word end.

Fig. 1 illustrates this concept for a three-word vocabulary (*A*, *B*, *C*), where the lexical tree is depicted in a simplified schematic form. In the set-up of Fig. 1, 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. To handle intraphrase silence models, we add a separate copy of the silence model (*Sil*) to each tree. In addition, we have a separate copy of the lexical tree for the first word in the sentence; this tree copy is given silence as predecessor word. As a result of this approach, the silence model copies do not require a special treatment, but can be processed like regular words of the vocabulary. However, there is one exception: at word boundaries, there is no language model probability for the silence models. As shown in Fig. 1, there are two types of path extensions and recombinations, namely in the interior of the words or lexical trees and at word boundaries. In the word interior, we have the bold lines representing the

transitions in the Hidden Markov models. At word boundaries, we have the thin and the dashed lines, which represent the bigram language model recombinations. Like the acoustic recombinations, they, too, are performed each time frame (10 ms). The dashed lines are related to recombinations for interphrase silence copies. To start up a new word hypotheses, we have to incorporate the bigram probabilities into the scores $Q_v(t,s=S_w)$ and to determine the best predecessor word v. This best score is then propagated into the root of the associated lexical tree, which is represented by the symbol $\square$. The symbol $\bigcirc$ denotes a word end.

For a quantitative specification of the search procedure, we assume that each arc of the lexical tree is represented by a Hidden Markov model. We will use the state index s directly and assume that the lexical structure is captured by the transition probabilities of the Hidden Markov model. To formulate the dynamic programming approach, we introduce the following two quantities (Ney, 1993):

$Q_v(t, s) :=$ overall score of the best partial path that at time t ends in state s of the lexical tree for predecessor v.
$B_v(t, s) :=$ starting time of the best partial path that at time t ends in state s of the lexical tree for predecessor v.

In other words $B_v(t, s)$ is the back pointer which points back to the starting time of the lexical tree copy for predecessor word v. This back pointer is needed because the definition of the score $Q_v(t, s)$ implies that the optimization over the unknown starting time of the lexical tree copy for predecessor word v has been carried out. Both quantities are evaluated using the dynamic programming recursion for $Q_v(t, s)$:

$$\begin{aligned} Q_v(t, s) &= \max_{\sigma}\{q(x_t, s|\sigma)\cdot Q_v(t-1, \sigma)\} \\ B_v(t, s) &= B_v(t-1, \sigma_v^{max}(t, s)), \end{aligned} \tag{1}$$

where $\sigma_v^{max}(t, s)$ is the optimum predecessor state for the hypothesis (t, s) and predecessor word v. $q(x_t, s|\sigma)$ is the product of transition and emission probabilities of the Hidden Markov models used for the context dependent or independent phonemes. The back pointers $B_v(t, s)$ are propagated according to the dynamic programming decision. Unlike the predecessor word v, the index w for the word under consideration is only needed and known when a path hypothesis reaches an end node of the lexical tree: each end node of the lexical tree is labeled with the corresponding word of the vocabulary.

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 v for each word w. To this purpose, we define:

$$H(w; t) := \max_{v}\{p(w|v)\cdot Q_v(t, S_w)\}, \tag{2}$$

where the state S_w denotes the terminal state of word w in the lexical tree. To propagate the path hypothesis into the lexical tree hypotheses or to start them up if they do not exist yet, we have to pass on the score and the time index *before* processing the hypotheses for time frame t:

TABLE II. One-pass algorithm ("single-best; lexical tree; bigram")

Proceed over time t from left to right
Acoustic level: process states of lexical trees
—Initialization: $Q_v(t-1,\ s=0)=H(v;\ t-1)$ $B_v(t-1,\ s=0)=t-1$
—Time alignment: $Q(t,\ s)$ using DP —Propagate back pointers $B_v(t,\ s)$
—Prune unlikely hypotheses —Purge bookkeeping lists
Word pair level: process word ends for each pair $(w;\ t)$ do
$H(w;\ t)=\max_v \{p(w\|v)\ Q_v(t,\ S_w)\}$
$v_0(w;\ t)=\arg\max_v \{p(w\|v)\ Q_v(t,\ S_w)\}$
—Store best predecessor $v_0:=v_0(w;\ t)$ —Store best boundary $\tau_0:=B_{v_0}(t,\ S_w)$

$$\begin{aligned} Q_v(t-1,\ s=0)&=H(v;\ t-1)\\ B_v(t-1,\ s=0)&=t-1. \end{aligned} \tag{3}$$

The details of the algorithm are summarized in Table II.

2.1.1. Garbage collection

For large-vocabulary recognition, it is essential to keep the storage costs as low as possible. To reduce the memory requirements back pointers and traceback arrays are needed, whereas the traceback arrays are used to record the decisions about the best predecessor word for each word start-up. At word boundaries, we store for each word end hypothesis: word index, ending time of the predecessor word, score and back pointer. The ending time of the predecessor word is not really needed, but useful for diagnostic purposes. During the recognition process, many of the hypothesis entries in the traceback arrays will become obsolete because their path extensions die out over time due to both the recombination and the pruning of hypotheses. In order to remove these obsolete hypothesis entries from the traceback arrays, we apply a garbage collection or purging method as follows. Each entry of the traceback array is extended by an additional component which is the so-called time stamp as suggested by Steinbiss (1992). Using the backpointers of the state hypotheses, we perform a traceback for each state hypothesis and mark the traceback entries reached with the current time frame as time stamp. Hence, all traceback entries that have a time stamp different from the current time frame can be re-used to store new hypotheses. Note that this garbage collection process is controlled using the *state hypotheses and reachable traceback entries* only so that the number of *dead* traceback entries does not matter. In principle, this

garbage collection process can be performed every time frame, but to reduce the overhead, it is sufficient to perform it in regular time intervals, say every 50th time frame.

2.1.2. Pruning techniques and language model look-ahead

Since full search is prohibitive, we use the time synchronous beam search strategy, where at each time frame only the most promising hypotheses are retained (Lowerre & Reddy, 1980). The pruning approach consists of three steps that are performed every 10-ms time frame (Steinbiss *et al.*, 1994).

- *Standard beam pruning* or so-called acoustic pruning is used to retain for further consideration only hypotheses with a score close to the best state hypothesis. Denoting the best scoring state hypothesis by

$$Q_{AC}(t) := \max_{(v,s)} \{Q_v(t, s)\},$$

we prune a state hypothesis $(s, t; v)$ if:

$$Q_v(t, s) < f_{AC} \cdot Q_{AC}(t).$$

The so-called beam width, i.e. the number of surviving state hypotheses, is controlled by the so-called acoustic pruning threshold f_{AC}.

- *Language model pruning* is applied only to tree start-up hypotheses as follows. At word ends hypotheses, the bigram probability is incorporated into the accumulated score, and the best score for each predecessor word is used to start up the corresponding tree hypothesis or is propagated into this tree hypothesis already exists. The scores of these tree start-up hypotheses are subjected to an additional pruning step:

$$Q_{LM}(t) := \max_v \{Q_v(t, s=0)\},$$

where $s=0$ is a fictitious state for initialization. Thus a tree start-up hypothesis is removed if:

$$Q_v(t, s=0) < f_{LM} \cdot Q_{LM}(t),$$

where f_{LM} is the so-called language model pruning threshold.

- *Histogram pruning* limits the number of surviving state hypotheses to a maximum number (*MaxHyp*). If the number of active states is larger than *MaxHyp*, only the best *MaxHyp* hypotheses are retained and the other hypotheses are removed. This pruning method is called histogram pruning because we use a histogram of the scores of the active states (Steinbiss *et al.*, 1994).

The efficiency of these pruning methods can be improved by using the so-called look-

TABLE III. Typical memory requirements for the search (20 000-word NAB task; bigram language model)

Type of array	Maximum number of entries	Number of bytes
State hypotheses	600 000	7 200 000
Arc hypotheses	200 000	1 600 000
Auxiliary arc hypotheses	65 000	1 040 000
Tree hypotheses	20 000	240 000
Traceback array	200 000	4 000 000
Total	1 085 000	13 820 000

ahead techniques, e.g. language model look-ahead (Odell *et al.*, 1994; Steinbiss *et al.*, 1994; Antoniol *et al.*, 1995; Renals & Hochberg, 1995; Alleva *et al.*, 1996; Ortmanns *et al.*, 1996*a*) and phoneme look-ahead (Ney *et al.*, 1992*a*; Haeb-Umbach & Ney, 1994). In this work, we employed only what we call unigram language model look-ahead (Steinbiss *et al.*, 1994) which works as follows. For each phoneme arc of the lexical tree, we consider the probabilities of a unigram language model in order to obtain an estimate of how likely we can reach an end node from the given phoneme arc. This anticipated language model probability is incorporated into the dynamic programming equation for computing $Q_v(t, s)$ each time a phoneme boundary is hypothesized. When reaching an end node, we then apply the true probability of the bigram language model after removing the estimate of the unigram language probability. The experimental tests show that this unigram language model look-ahead reduces the search effort by a factor of 4 in combination with the above pruning techniques (Steinbiss *et al.*, 1994; Ortmanns *et al.*, 1996*a*).

2.1.3. Dynamic search space and memory requirements

In this subsection, we consider the memory requirement for the word conditioned tree search method of the present implementation. As we will see later, these memory requirements will not be increased by the algorithm for word graph construction.

The implementation is based on lists of active hypotheses for states, arcs and trees; these lists are implemented using static arrays (Ney *et al.*, 1992*b*; Haeb-Umbach & Ney, 1994). Using these lists of active hypotheses, the search space is constructed dynamically during the process of recognition. Although the details of this implementation are out of the scope of this paper, we give a short summary of the memory requirements. Each entry of the state array consists of three components, namely score, back pointer and state index, for which all in all 12 bytes are used per entry. The memory for the arc organization is divided into two parts, one for representing the arc hypotheses of all active trees (8 bytes per entry) and another auxiliary one for storing the arc hypotheses during the process of recombining across-phoneme hypotheses within a given tree (16 bytes per entry). For the tree hypotheses, we always use the maximum number as array size, which is the vocabulary size; each entry of the tree hypothesis consists of 12 bytes (tree identifier, pointer into itself and pointer into the list of arc hypotheses). In addition, we use a traceback array to store the information at phrase level including word index, score, back pointer, ending time and time stamp for each surviving word hypothesis, which requires 20 bytes per entry.

Table III gives an overview of the resulting memory costs for the NAB 20 000-word

task used in the experiments. For this task, the tree representation of the pronunciation lexicon consists of about 65 000 phoneme arcs. The maximum number of hypotheses is limited to 600 000 states and 200 000 arcs, respectively.

2.2. Extension to trigram language models

So far, we have considered the one-pass search approach only in the context of a bigram language model. To extend the word conditioned tree search method from a bigram to a trigram language model, we have to take into account that for a trigram the language model probabilities are conditioned on the previous two words rather than one predecessor word in the bigram case (Ney, 1993; Odell *et al.*, 1994; Ortmanns *et al.*, 1996*b*). Therefore, the incorporation of a trigram into the word conditioned tree search method requires a restructuring of the search space organization. Fig. 2 illustrates the search space using a trigram model. For each two-word history (u, v), we introduce a separate copy of the lexical tree; in Fig. 2, the root of each tree copy is labeled with its two-word history. As in the case of a bigram language model, the structure of the search space is defined in such a way that in the search network the probabilities or costs of each edge depend *only* on the edge itself (along with its start and end vertex) and nothing else. This property of the search network allows us to directly apply the principle of dynamic programming. Note that in comparison with a bigram organized search space, the size of the potential search space is increased drastically by an additional factor, which is the vocabulary size. Hence, in order to keep the search effort manageable, an efficient pruning strategy as described before is even more crucial for the case of a trigram language model.

For simplicity, in Fig. 2, we omit the silence copies. To allow for intraphrase silence, we use the same concept as for the bigram language model (Ney *et al.*, 1992*a*; Haeb-Umbach & Ney, 1994). For the trigram language model recombinations, we need the identity of the two non-silence predecessor words, and therefore we need a separate copy of the silence model for each pair of non-silence predecessor words.

To formulate the dynamic programming approach, we introduce the key quantities $Q_{uv}(t, s)$ and $B_{uv}(t, s)$ which must now depend on the two predecessor words (u, v) to allow for the application of the trigram language model:

> $Q_{uv}(t, s) :=$ overall score of the best path up to time t that ends in state s of the lexical tree for the two-word history (u, v).
> $B_{uv}(t, s) :=$ starting time of the best path up to time t that ends in state s of the lexical tree for the two-word history (u, v).

As mentioned in the case of the bigram search, $B_{uv}(t, s)$ is the back pointer which depends on the starting time of the copy of the lexical tree for the two predecessor words (u, v), but *not* on the ending time of the two-word history (u, v).

The dynamic programming recursion within each copy of the lexical tree remains the same as for the bigram case. So we have the usual dynamic programming recursion for the word interior:

$$\begin{aligned} Q_{uv}(t, s) &= \max_{\sigma} \{q(x_t, s|\sigma) \cdot Q_{uv}(t-1, \sigma)\} \\ B_{uv}(t, s) &= B_{uv}(t-1, \sigma_{uv}^{max}(t, s)), \end{aligned} \tag{4}$$

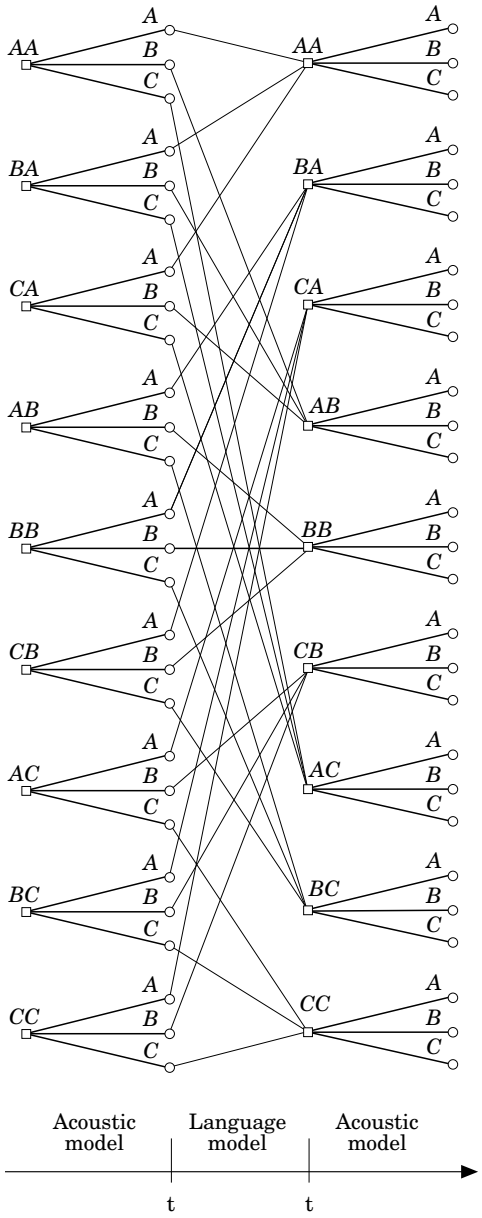

Figure 2. Trigram language model recombination for a tree lexicon (three-word vocabulary: *A*, *B*, *C*) without intraphrase silence handling. *BA*: Tree copy of the two-word history (*B*, *A*); ——: acoustic recombination within a tree copy; ——: trigram language model recombination; □: root of a tree copy; ○: word end.

where $\sigma_{uv}^{max}(t, s)$ is the optimum predecessor state for the hypothesis (t, s) and two-word history (u, v). As before, $q(x_t, s|\sigma)$ is the product of transition and emission probabilities. To perform the recombination across the word boundaries, we define the quantity:

TABLE IV. One-pass algorithm ("single best") using a trigram language model

Proceed over time t from left to right

Acoustic level: process states of lexical trees

—Initialization: $Q_{uv}(t-1,\ s=0)=H(u,\ v;\ t-1)$
$B_{uv}(t-1,\ s=0)=t-1$

—Time alignment: $Q_{uv}(t,\ s)$ using DP
—Propagate back pointers $B_{uv}(t,\ s)$

—Prune unlikely hypotheses
—Purge bookkeeping lists

Word pair level: process word ends

For each triple $(v,\ w;\ t)$ do

$$H(v,\ w;\ t)=\max_{u}\{p(w|u,\ v)\ Q_{uv}(t,\ S_w)\}$$

$$u_0(v,\ w;\ t)=\arg\max_{u}\{p(w|u,\ v)\ Q_{uv}(t,\ S_w)\}$$

—Store best predecessor $u_0 := u_0(v,\ w;\ t)$
—Store best boundary $\tau_0 := B_{u_0}(t,\ S_w)$

$$H(v,\ w;\ t) := \max_{u}\{p(w|u,\ v)\cdot Q_{uv}(t,\ S_w)\}, \tag{5}$$

where $p(w|u,\ v)$ is the conditional trigram probability for the word triple $(u,\ v,\ w)$. As in the case of the bigram language model, to start up the lexical tree hypotheses, we have to pass on the score and the time index *before* processing the hypotheses for time frame t:

$$\begin{aligned} Q_{uv}(t-1,\ s=0)&=H(u,\ v;\ t-1)\\ B_{uv}(t-1,\ s=0)&=t-1, \end{aligned} \tag{6}$$

where we have introduced the fictitious state $s=0$ for initialization. Table IV shows the details of the algorithm.

To handle the search efficiently, we have to identify the tree hypotheses by their two-word history. In the case of a bigram language model, an array-based method with indirected pointers (Ney *et al.*, 1992*b*) is used. Due to the size of the array needed, namely $20\,000^2$ for a 20 000-word vocabulary, this method is not viable for a trigram language model. Instead, a hashing approach is used (Ortmanns *et al.*, 1996*b*). The index of the hash table is computed from a bijective function of the word pair index (u,v), e.g. $f(v,\ w)=W\cdot v+w$, where W is the vocabulary size. This hashing method was found to cause only negligible overhead.

3. Word graph method

3.1. Word graph specification

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:

> Hypothesizing a word w and its 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 edge 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. In the one-pass algorithm for computing the single-best sentence, we have computed the hypotheses in a time-synchronous fashion and have propagated the hypotheses from left to right over the time axis. We will use the same principle of time synchrony for the word graph generation. To this purpose, we introduce the following definitions.

$h(w; \tau, t) := Pr(x_{\tau+1}^{t}|w) =$ probability that word w produces the acoustic vectors $x_{\tau+1} \ldots x_t$.

$G(w_1^n;\ t) := Pr(w_1^n) \cdot Pr(x_1^t|w^n) =$ (joint) probability of generating the acoustic vectors $x_1 \ldots x_t$ and a word sequence $w_1 \ldots w_n$ 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.

$$\begin{array}{l} x_1, \ldots, \ldots x_\tau\ x_{\tau+1}, \ldots, x_t\ x_{t+1}, \ldots, \ldots, x_T \\ G(w_1^{n-1}; \tau)\quad h(w_n; \tau, t) \qquad \ldots \end{array}$$

From this decomposition, it is clear that the score $G(w_1^n;\ t)$ can be computed from the score $G(w_1^{n-1};\ t)$ by optimizing over the unknown word boundary τ:

$$G(w_1^n;\ t) = \max_\tau \{Pr(w_n|w_1^{n-1}) \cdot G(w_1^{n-1};\ \tau) \cdot h(w_n; \tau, t)\} \tag{7}$$

$$= Pr(w_n|w_1^{n-1}) \cdot \max_\tau \{G(w_1^{n-1};\ \tau) \cdot h(w_n; \tau, t)\}, \tag{8}$$

where we have used the *conditional* probability $Pr(w_n|w_1^{n-1})$ of the language model. To construct a word graph, we introduce a formal definition of the word boundary $\tau(w_1^n;\ t)$ between the word hypothesis w_n ending at time t and the predecessor sequence hypothesis w_1^{n-1}:

$$\tau(t; w_1^n) := \arg\max_{\tau}\{G(w_1^{n-1}; \tau)\cdot h(w_n; \tau, t)\}. \tag{9}$$

It should be emphasized that the language model probability does *not* affect the optimal word boundary according to Equation (8) and is therefore omitted in the definition of the word boundary function $\tau(w_1^n; t)$. Thus far we have considered the most general case in two aspects: first, the word boundary function was not constrained in any way. Second, the language model was not constrained in any way. We will first narrow down the language model to the widely used m-gram language models and come back to the word boundary function later.

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 w_1^n only by their final words $u_2^m := w_{n-m+2}^n$. The corresponding score is denoted by $H(u_2^m; t)$ and is defined as the joint probability of generating the acoustic vectors $x_1 \ldots x_t$ and a word sequence with *ending* sequence u_2^m and *ending* time t:

$$H(u_2^m; t) := \max_{w_1^n}\{Pr(w_1^n)\cdot Pr(x_1^t|w_1^n): w_{n-m+2}^n = u_2^m\}, \tag{10}$$

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:

$$H(u_2^m; t) = \max_{u_1} \hat{H}(u_2^m; t) \tag{11}$$

$$\text{with } \hat{H}(u_1^m; t) := p(u_m|u_1^{m-1})\, H(u_1^{m-1}; \tau(t; u_1^m))\, h(u_m; \tau(t; u_1^m), t). \tag{12}$$

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 strategy. For the word boundary itself, we have to use the quantity $H(u_2^m; t)$ rather than $G(w_1^n; t)$:

$$\tau(t; u_1^m) := \arg\max_{\tau}\{H(u_1^{m-1}; \tau)\, h(u_m; \tau, t)\}. \tag{13}$$

3.2. Word pair approximation

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 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 between words u_{m-1} and u_m (Schwartz & Austin, 1991). This so-called word pair approximation is

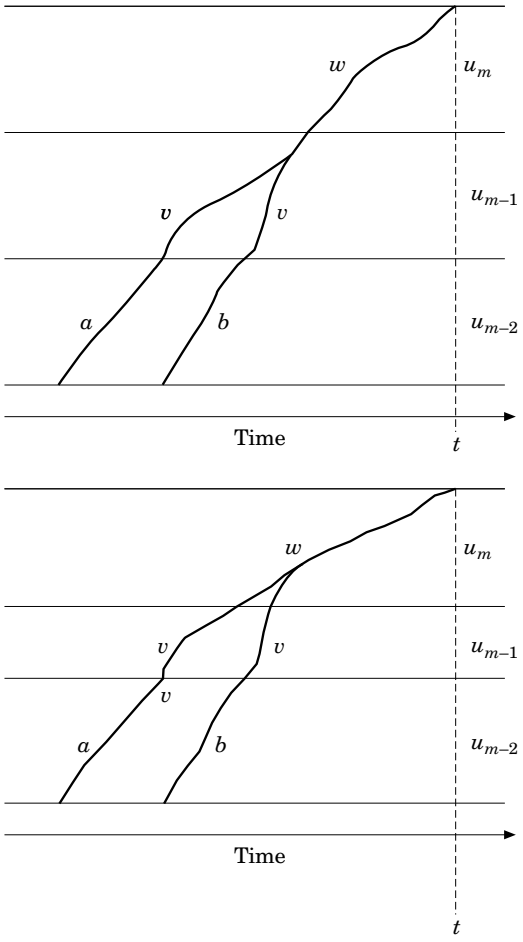

Figure 3. Illustration of the word pair approximation for two cases: (a) good example: the predecessor word $u_{m-1} := v$ of word $u_m := w$ is sufficiently long; (b) bad example: the predecessor word $u_{m-1} := v$ of word $u_m := w$ is too short.

illustrated in Fig. 3. For a word hypothesis w and an ending time hypothesis t, this figure shows the time alignment path for the word $w = u_m$ itself and its predecessor words u_{m-2}^{m-1} to illustrate the definition of the word boundary $\tau(t; u_1^m)$. In general, this boundary, i.e. the start time word w as given by time alignment, will depend on the immediate predecessor word $u_m = v$. The question of whether this dependence reaches beyond the immediate predecessor word is illustrated by showing a good [Fig. 3(a)] and a bad [Fig. 3(b)] example. For simplification, we have assumed that the reference models of the predecessor words $u_m = a$ and $u_m = b$ have the same length. 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. In formulae, we express the word pair approximation by the equation:

$$\tau(t; u_1^m) = const(u_1^{m-2}) \text{ or } \tau(t; u_1^m) = \tau(t; u_{m-1}^m), \tag{14}$$

i.e. the word boundary function does *not* depend on u_1^{m-2}. Assuming the word pair approximation, we have the following algorithm for word graph construction:

- At each time frame t, we consider all word pairs $u_{m-1}^{m} = (v, w)$. Using a beam search strategy, we will limit ourselves to the most probable word pairs.
- For each triple $(t; v, w)$, we have to keep track of:
  —the word boundary $\tau(t; v, w)$
  —the word score $h(w; \tau(t; v, w), t)$.
- At the end of the speech signal, the word graph is constructed by tracing back through the bookkeeping lists.

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 Steinbiss (1991) to produce a list of n-best sentences.

3.3. Word graph generation algorithm

The computation of 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 (Sakoe, 1979) or the one-pass algorithm described 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 word conditioned variant of the one-pass algorithm as presented in the preceding section.

To extend the one-pass word 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:

$$\tau(t; v, w) = B_v(t, S_w).$$

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

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

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

$$H(w; t) = \max_v \{p(w|v) \cdot Q_v(t, S_w)\}. \tag{15}$$

TABLE V. One-pass algorithm for word graph construction ("single best" and "word graph")

Proceed over time t from left to right

Acoustic level: process states of lexical trees

—Initialization: $Q_v(t-1,\ s=0)=H(v;\ t-1)$
$B_v(t-1,\ s=0)=t-1$

—Time alignment: $Q_v(t,\ s)$ using DP
—Propagate back pointers $B_v(t,\ s)$

—Prune unlikely hypotheses
—Purge bookkeeping lists

Word pair level: process word ends

Single-best: for each pair $(w;\ t)$ do

$$H(w;\ t)=\max_v\{p(w|v)\ Q_v(t,\ S_w)\}$$

$$v_0(w;\ t)=\arg\max_v\{p(w|v)\ Q_v(t,\ S_w)\}$$

—Store best predecessor $v_0 := v_0(w;\ t)$
—Store best boundary $\tau_0 := B_{v_0}(t,\ S_w)$

Word graph: for each triple $(t;\ v,\ w)$ store

—Word boundary $\tau(t;\ v,\ w) := B_v(t,\ S_w)$
—Word score $h(w;\ \tau,\ t) := Q_v(t,\ S_w)/H(v;\ \tau)$

Phrase level search (optional)

The details of the algorithm are summarized in Table V. 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 for the final recognition. Depending on whether the phrase-level recognition is carried out in a time-synchronous fashion or not, we can distinguish the following two strategies in using a trigram or higher m-gram language model.

- *Extended one-pass approach*: the word pair approximation serves only as a simplification in the one-pass strategy in order to avoid the large number of copies of the lexical tree as required by the language model.
- *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.

From the concepts developed so far, it should be obvious that there is only a gradual difference between these two strategies.

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.

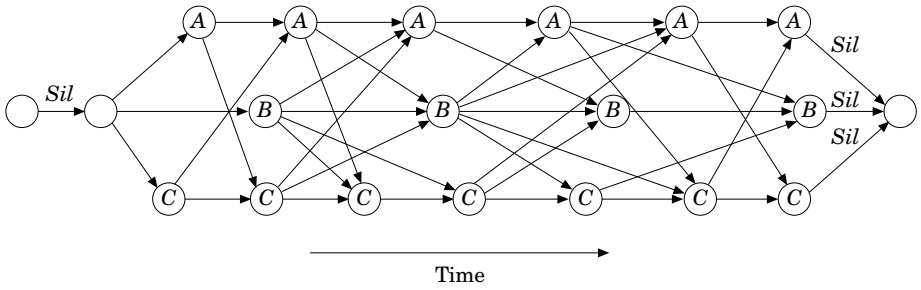

Figure 4. Example of a word graph (three-word vocabulary: *A*, *B*, *C*).

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. 4. The edges 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.

- There is a maximum for the number of incoming word edges in any node, namely the vocabulary size which is the maximum number of possible predecessor words.
- There is no maximum for the number of outgoing word edges; this effect is due to the fact that, even for the same predecessor word, a word can have different ending time hypotheses.

There are two refinements of the word graph method which suggest themselves: (1) For short words or function words like articles and prepositions, the quality of the word pair approximation might be questionable, and word triples or higher word *m*-tuples might be used instead in these cases. (2) 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.

So far, we have not considered the handling of intraphrase silence in the presentation of the algorithm. As in the case of the bigram language model, we make the silence hypotheses dependent on the (non-silence) predecessor words. The effect of these silence copies on the word graph construction is illustrated in Fig. 5. A word v and the associated silence copy ${}_{v}Sil$ are treated as one single unit. However, due to this trick, the information of whether there has been a silence portion ${}_{v}Sil$ between the word w and the predecessor word v is lost and cannot be recovered easily.

3.4. Word graph pruning

The word graph generated by the acoustic recognition process can be very large. To reduce the size of the word graph, pruning methods can be used in a similar manner as described in Section 2.1.2 without significantly increasing the word error rate. During the acoustic recognition process, only the most likely word hypotheses (w; t) have to be retained at every time frame t. Therefore, we have to determine the score of the best word hypothesis $H_{max}(t)$ as follows:

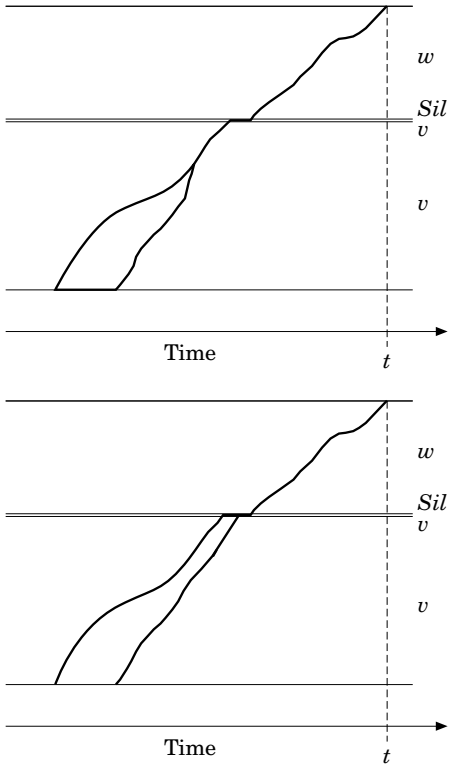

Figure 5. Intraphrase silence handling in the word graph construction.

$$H_{max}(t) = \max_{w}\{H(w;\, t)\}, \tag{16}$$

where $H(w;\, t)$ is:

$$H(w;\, t) = \max_{v}\{p(w|v) \cdot H(v;\, \tau(t;\, v,\, w)) \cdot h(w;\, \tau(t;\, v,\, w),\, t)\}. \tag{17}$$

The pruning of the word graph is based on the usual concept of beam search: hypotheses $(w;\, t)$ with a score relatively close to best word hypothesis are retained as active, the others are pruned. In formulae, we have:

$$p(w|v) \cdot H(v;\, \tau(t;\, v,\, w)) \cdot h(w;\, \tau(t;\, v,\, w),\, t) < f_{LAT} \cdot H_{max}(t). \tag{18}$$

$f_{LAT} < 1$ denotes the word graph pruning threshold. This pruning method can also be combined with histogram pruning to limit the maximum number of word hypotheses per time frame. In order to keep the storage costs for the word graph small, a garbage collection is performed after pruning in a similar way as described in a previous section.

TABLE VI. Word graph rescoring: search algorithm at phrase level (second pass) using an m-gram language model

Input: set of word boundaries $\tau(t; u_1^m)$
and of word scores $h(u_m; \tau, t)$
Proceed over time t from left to right
Process each word pair u_m^{m-1} in the graph
Get word boundaries $\tau(t; u_1^m)$ and scores $h(u_m; \tau, t)$
Process each word sequence u_1^m
$\hat{H}(u_1^m; t) := p(u_m|u_{1-1}^m) \cdot H(u_1^{m-1}; \tau) \cdot h(u_m; \tau, t)$
$H(u_2^m; t) = \max_{u_1} \hat{H}(u_1^m; t)$
$B(u_2^m; t) = \operatorname{argmax}_{u_1} \hat{H}(u_1^m; t)$
Traceback: use back pointers $\{B(u_2^m; t)\}$

3.5. Word graph rescoring

Given a word graph and an m-gram language model, the second-pass of the word graph method can be carried out at the word level using a left-to-right dynamic programming algorithm as given by Equation (12). The implementation of this algorithm is fairly straightforward. Of course, the list (or set) of hypothesized word sequences u_1^m should be represented in an efficient way. The algorithm for the search through the graph are summarized in Table VI. The cost of this search through the word graph depends on the type and complexity of the language model. This second step of the recognition operation is often referred to as rescoring. As we will show in the experiments, for a trigram language model as it is used in the tests on the NAB task, the cost of the rescoring is typically less than 1% of the effort for constructing the word graph (using a bigram language model).

4. Experimental results

4.1. Recognition task and database

The experimental tests were carried out on the ARPA North American Business (NAB'94) H1 development corpus including 310 sentences with 7387 words by 10 male and 10 female speakers. The number of spoken words which were out-of-vocabulary words relative to the 20 000-word vocabulary was 199. The training of the emission probability distributions of the underlying Hidden Markov models was performed on the so-called *WSJ*0 and *WSJ*1 training data as described in Dugast *et al.* (1995). In the experiment we used about 290 000 Laplacian mixture densities (with a single pooled vector of absolute deviations) for each gender. In all the recognition experiments, we

TABLE VII. Some statistics about the NAB'94 H1 development test set (OOV = out-of-vocabulary)

Vocabulary size	Spoken words	OOV words	Test set perplexity	
			Bigram	Trigram
19 977	7387	199	198·1	130·2

used the official 20 000-word trigram language model for the NAB'94 task (Rosenfeld, 1995). The test conditions are summarized in Table VII.

4.2. Results for the word graph method

In this subsection, we report on recognition experiments using the word graph method. To give a quantitative specification of the size of the word graphs used and the recognition results, we introduce the following terminology:

- For a spoken sentence, the *word graph density* (WGD) is defined as the total number of word graph edges divided by the number of actually spoken words. Similarly, the *node graph density* (NGD) and the *boundary graph density* (BGD) denote the number of nodes and of different word boundaries, respectively, per spoken word.
- The *graph word error rate* (GER) is computed 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 (SUB), deletions (DEL) and insertions (INS). This measure provides a lower bound of the word error rate for this word graph and gives a better measurement of the word graph quality than the graph sentence error rate. The graph word error rate (GER) is to be distinguished from the standard *recognition word error rate* (WER).

To study the effect of the word graph size on the recognition performance, a conservatively large word graph was computed using the word graph algorithm described and stored on disk for each test sentence. The word graph had been constructed using a bigram language model with a test set perplexity of $PP_{bi} = 198{\cdot}1$. On average, the search space consisted of 27 672 active states, 7674 active arcs and 115 active trees per time frame during the first pass of the two-pass search strategy and results in a word error rate of 16·5%. When generating the word graph, the maximum number of word end hypotheses per time frame was limited to 1000. Then by varying the so-called word graph pruning threshold f_{LAT}, the size of the word graph was reduced and the effect on the graph word error rate and the recognition word error rate was measured. For the recognition test, a full search through the word graph was performed using a trigram language model (perplexity of 130·2), i.e. no pruning was applied. In all tests, we used only a scaling factor for the logarithms of the language model probabilities; there was no word penalty used although it could slightly decrease the word error rate. For

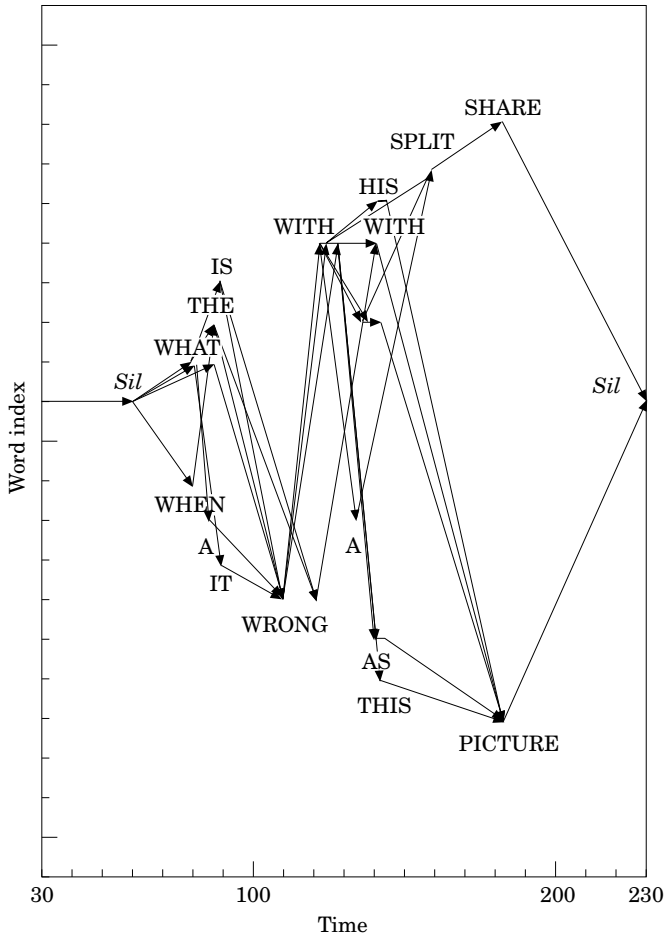

Figure 6. Word graph for the spoken sentence: WHAT IS WRONG WITH THIS PICTURE?

illustration purposes, two examples of word graphs are shown in Figs 6 and 7 as computed on the corpus. The layout of the word graphs is the same as that in Fig. 4: the horizontal axis is the time axis; along the vertical axis, the figures show the various word hypotheses in the word graph. Note that, due to the definition of the vertical axis, vertical distances do *not* indicate any acoustic similarities between words.

Table VIII summarizes the results for the obtained for the word graph method under the above conditions. In the actual implementation, we use the logarithms of the probabilities rather than the probabilities themselves; therefore, Table VIII reports the values of the logarithm F_{LAT} (in relative units) of the pruning threshold f_{LAT}. For various values of the pruning threshold, the table shows the word graph size in terms of word graph density (WGD), number of word graph nodes (NGD) and number of word boundaries (BGD). In addition, the table reports the graph word error rate and the recognition word error rate, both of which are given in terms of word deletions, insertions and substitutions. For the smallest word graph with WGD = 1·26, there is

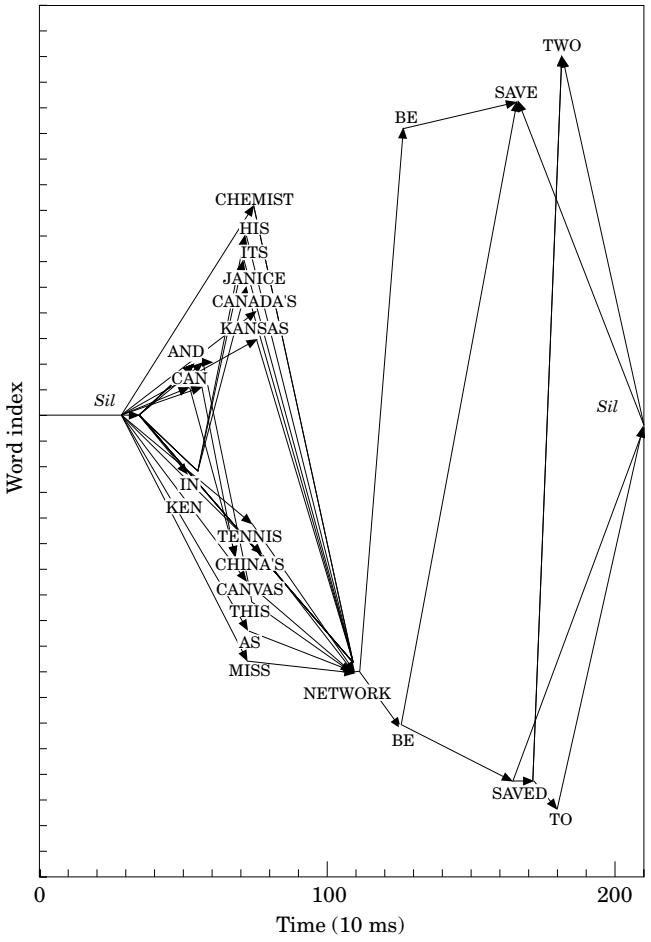

Figure 7. Word graph for the spoken sentence: CAN THIS NETWORK BE SAVED?

virtually no difference between the graph word error rate of 16·3% and the recognition word error rate of 16·4%. These error rates are close to the bigram word error rate of 16·5% as it must be the case. By using the largest word graph with WGD = 1467·21, the recognition error rate is reduced down to 14·3%, whereas the graph word error rate is affected much more: it goes down from 16·3% to 4·2%. Here, we have to keep in mind that the two error rates cannot be decreased below 2·7%, the out-of-vocabulary rate. As far as the recognition word error rate is concerned, we see that the minimum of 14·3% is already achieved for a word graph density (WGD) of 10·67. In contrast with this result, for the graph word error rate, there is a steady improvement down to 4·2% by increasing the word graph density. Evidently the explanation is that the acoustic and language model probabilities are not good enough to achieve the possible minimum of 4·2%.

To illustrate the usefulness of the word graph approach, an additional recognition experiment was carried out using a language model that has been extended by a cache

TABLE VIII. Recognition results for the word graph method (trigram language model with PP_{tri} = 130·2; OOV rate: 2·7%; word graph generation: 27 672 states, 7674 arcs, 115 trees)

	Graph density			Graph word error rate		Recognition word error rate	
F_{LAT}	WGD	NGD	BGD	DEL–INS–SUB	GER [%]	DEL–INS–SUB = TOTAL	WER [%]
300	1476·21	181·80	19·67	9– 38–262	4·2	120–202–733 = 1055	14·3
150	1415·89	175·93	19·25	9– 38–262	4·2	120–202–733 = 1055	14·3
100	684·47	104·22	14·50	13– 38–263	4·3	120–202–733 = 1055	14·3
90	460·42	77·35	12·31	12– 38–270	4·3	120–202–733 = 1055	14·3
80	269·17	51·75	9·84	15– 44–272	4·5	120–202–733 = 1055	14·3
70	137·39	31·43	7·42	19– 51–282	4·8	120–202–733 = 1055	14·3
60	60·15	17·20	5·33	23– 56–301	5·1	122–201–730 = 1053	14·3
50	25·22	8·98	3·77	40– 64–325	5·8	124–200–732 = 1056	14·3
40	10·67	4·79	2·66	50– 81–368	6·8	122–202–735 = 1059	14·3
30	4·53	2·75	2·01	84– 94–445	8·4	135–191–754 = 1080	14·6
20	2·40	1·83	1·60	113–129–543	10·6	147–184–762 = 1093	14·8
10	1·56	1·41	1·36	156–160–700	13·8	173–180–801 = 1154	15·6
5	1·36	1·29	1·28	168–177–788	15·3	181–190–831 = 1202	16·3
1	1·26	1·25	1·24	167–192–845	16·3	167–195–851 = 1213	16·4

Abbreviations: WGD: word graph density; NGD: number of word graph nodes; BGD: number of word boundaries; DEL: word deletions; INS: word insertions; SUB: word substitutions; WER: word error rate.

TABLE IX. Recognition results for the word graph method using a unigram/bigram cache model

Language model		PP	DEL–INS–SUB = TOTAL	WER [%]
Bigram:	without cache	198·1	179–195–846 = 1220	16·5
	with cache: *unsupervised*	178·0	190–179–835 = 1204	16·3
	supervised	171·2	194–179–806 = 1179	16·0
Trigram:	without cache	130·2	120–202–733 = 1055	14·3
	with cache: *unsupervised*	117·1	127–182–713 = 1032	14·0
	supervised	113·2	128–194–699 = 1021	13·8

Abbreviations: PP: perplexity; DEL: word deletions; INS: word insertions; SUB: word substitutions; WER: word error rate.

model. The so-called cache model can be considered as a short-term memory where the probability of the most recent unigrams and bigrams is increased (Kuhn & De Mori, 1990; Generet *et al.*, 1995). In the recognition experiments, we combined the baseline bigram/trigram model with this cache model. The cache was used as follows. The cache was initialized, i.e. reset, each time a new speaker started. After each recognition of a sentence, the cache was updated. This was performed in two variants, namely unsupervised and supervised. In the unsupervised variant, the recognized words were used to fill the memory of the cache model. In the supervised variant, the cache was updated using the actually spoken words. Limiting the number of words in the cache to the most recent M words did not improve the perplexity; so the cache as it was used here is simply a memory of all previous words, either spoken or recognized. The perplexities PP and recognition results are shown in Table IX. In the bigram case,

TABLE X. Recognition results for the integrated method (trigram language model with $PP_{tri} = 130{\cdot}2$) as a function of the acoustic pruning threshold f_{AC}

f_{AC}	Average number of active				DEL–INS–SUB = TOTAL	WER [%]
	States	Arcs	Trees	Word ends		
80	4714	1392	29	80	128–211–757 = 1096	14·8
100	18 734	5430	70	294	120–206–721 = 1047	14·2
120	48 940	13 877	112	702	120–204–717 = 1041	14·1
130	67 541	18 764	130	953	120–201–712 = 1033	14·0
140	86 772	23 688	145	1223	121–200–709 = 1030	13·9

Abbreviations: DEL: word deletions; INS: word insertions; SUB: word substitutions; WER: word error rate.

the word error can be decreased by the use of the cache model from 16·5% to 16·3% and 16·0% for the unsupervised and the supervised variant, respectively. When using a trigram language model, the cache model results in a similar improvement. All these improvements tend to be small, which is a natural result of the fact that the cache model leads only to a slight improvement of the perplexity (see Table IX). Nevertheless, there is a *consistent* improvement in the word error rates, which result in another example of the usefulness of the word graph method.

4.3. Results for the integrated method

To verify the viability and the quality of the word graph method, in particular of the word pair approximation, we carried out another series of experiments, in which we used the integrated search method in combination with a trigram language model. In particular, we investigated the recognition accuracy as a function of the search effort by varying the acoustic pruning threshold f_{AC}, while keeping the other two pruning parameters f_{LM} and *MaxHyp* fixed. Table X shows the results of the recognition experiments. In addition to the word errors, Table X shows the search space, which is given in terms of the average number (per time frame) of active states, of active arcs, of active trees and of active word ends. From the table, it can be seen that by increasing the average number of active states per time frame from 4714 to 48 940 the word error rate is reduced from 14·8% to 14·1%. To further reduce the word error rate to 13·9 we have to double the average number of state hypotheses. Nevertheless, the actual search space is still manageable and is nine orders of magnitude lower than the potential size of the search space which is

$$20\,000^2 \text{ trees} \cdot 65\,000 \text{ arcs/tree} \cdot 6 \text{ states/arc} = 1{\cdot}56 \times 10^{14} \text{ states}.$$

This result demonstrates the efficiency of the beam search concept in the integrated search method.

4.4. Comparison: word graph method vs. integrated method

The results of the word graph method will now be compared with the results of the integrated search method. To this purpose, we consider the best results obtained for

Table XI. Comparison of the integrated method with the word graph method (NAB'94 H1 development set, 310 sentences = 7387 words)

		Word errors
Number of:	Sentences	Integrated/word graph
With identical score	277	898/898
With better score	32	121/146
With worse score	1	11/11
Total	310	1030/1055

Table XII. Three test sentences, for which the integrated method worked better than the word graph method (NAB'94 H1 development set, 310 sentences = 7387 words)

Spoken sentence:	… A FEW MUTUAL FUND INVESTORS *INTO* MODELS OF …
Word graph method:	… A FEW MUTUAL FUND INVESTORS *IN TWO* MODELS OF …
Integrated method:	… A FEW MUTUAL FUND INVESTORS *INTO* MODELS OF …
Spoken sentence:	… WHEN YOU BUY A MUTUAL FUND FROM *YOUR* BROKER
Word graph method:	… WHEN YOU BUY A MUTUAL FUND FROM *A* BROKER
Integrated method:	… WHEN YOU BUY A MUTUAL FUND FROM *YOUR* BROKER
Spoken sentence:	… I ALMOST WISH I HADN'T SEEN *THIS* PART
Word graph method:	… I ALMOST WISH I HADN'T SEEN *AS* PART
Integrated method:	… I ALMOST WISH I HADN'T SEEN *THIS* PART

each of the two methods in the case of a trigram language model (see Tables VIII and X). The integrated method produces 1030 word errors in comparison with 1055 word errors obtained for the word graph method. However, we have to take into account that search and recognition errors are not the same thing. Therefore, we have compared the recognition score of each sentence for the two methods. The results are summarized in Table XI. Out of the 310 test sentences, there were 277 sentences with *identical* scores for the two methods. There were 32 sentences for which the integrated method produced a better score than the word graph method, and there was only one sentence for which the opposite result was true. To see in which cases the word graph approach runs into problems, we selected three sentences for which the integrated method produced less errors than the word graph method. Table XII shows these three sentences with the recognition results for the integrated method and for the word graph method. As can be expected, the word graph method seems to run into problems for short words. In the examples shown in Table XII, the recognition errors are related to the word pairings "*in two–into*", "*your–a*", "*this–as*".

The overall result of the comparison of the word graph method with the integrated method can be summarized as follows. For about 90% of the sentences, there was no

TABLE XIII. Computational effort for the word graph method and the integrated method

	Word graph method		Integrated method	
Search space: states/arcs/trees	16274/4516/41		32291/9413/108	
Word error rate	81/519 = 15·6%		80/519 = 15·4%	
Time	(s)	(%)	(s)	(%)
Log-likelihood computation	14 830	78·2	15 174	67·2
Acoustic search	3603	19·0	6576	29·1
Lang. model recombination	139	0·7	621	2·7
Word graph rescoring	210	1·1	—	—
Other operations	177	1·0	208	1·0
Overall recognition	18 959	100·0	22 579	100·0
Real time factor	90	—	107	—

Subset of NAB'94 H1 development set: 20 speakers (10 male and 10 female speakers), 20 sentences = 519 spoken words (21 OOV words) = 211 s, $PP_{bi} = 201{\cdot}3$, $PP_{tri} = 134{\cdot}1$.
The experiments were run on an SGI workstation (Indy R4400, SpecInt'92: 94).

degradation in terms of search errors by the word graph method. Considering the recognition errors, the result is even more remarkable: there was only a relative increase by 2·4% in the word error rate over the best result of the integrated method, namely from 1030 to 1055 word errors. To obtain these error rates, the search effort in terms of state hypotheses was increased by more than a factor of 3 for the integrated method in comparison with the word graph method. At the same time, we have to keep in mind that the integrated method does not offer the flexibility of the word graph method. In another experiment (Aubert & Ney, 1995), the word graph method has been tested successfully on a 64 000-word task. All these experiments indicate that the word pair approximation works very well. Even for very short predecessor words, there are only a few exceptional cases in which the word pair approximation deteriorates the recognition performance in comparison with the much more costly integrated search method.

Finally, we will consider and compare a breakdown of the computational cost of the two methods. To measure the computational cost of the recognition, the exact times of the various operations were measured for recognition tests on a subset of the NAB'94 H1 development corpus which contained every first sentence of the 20 speakers. The results are shown in Table XIII. The computation times are given for each of the three operations: log-likelihood calculation, acoustic search and the language model recombinations. Due to the high cost of the log-likelihood calculation, the overall result differs only by about 25% for the two methods. The cost for the search itself, i.e. the time for the acoustic search and the language model recombinations, is nearly halved by the word graph method. Therefore, in cases where the computation time is an issue, a separate processor for calculating the log-likelihoods is highly desirable. Obviously, the word graph rescoring is applied only in the case of the word graph method and requires only 1·1% of the overall time. For the experiments, which were performed on an SGI workstation Indy R4400, the response times were 90 and 107 times real time for the word graph method and the integrated method, respectively.

To reduce the computational cost of the log-likelihood calculations, fast techniques for the log-likelihood calculations (Bocchieri, 1993; Beyerlein & Ullrich, 1995) are currently being studied. In addition, the search effort can be further decreased by using the so-called look-ahead techniques, both at the acoustic–phonetic level, e.g. a phoneme look-ahead (Haeb-Umbach & Ney, 1994), and in the context of the bigram language model (Odell *et al.*, 1994; Renals & Hochberg, 1995; Alleva *et al.*, 1996; Ortmanns *et al.*, 1996*a*).

5. Summary

This paper has presented a consistent and formal framework for the definition and generation of word graphs for large-vocabulary continuous-speech recognition. It has been shown that the resulting algorithm can be formulated as an extension of the single-best one-pass dynamic programming algorithm and results in a very straightforward and efficient implementation. In addition, we have used the so-called word pair approximation in the construction of word graphs and studied its viability in recognition experiments. These experiments on the NAB'94 20 000-word task have demonstrated that the word graph method in combination with the word pair approximation is able to produce very high quality word graphs. When using a trigram language model, the word graph method performed virtually as well as the fully integrated method, which is less flexible and computationally much more expensive.

The work carried out at RWTH Aachen was in part supported by Philips Research Laboratories Aachen.

References

Alleva, F., Huang, X. & Hwang, M.-Y. (1993). An improved search algorithm using incremental knowledge for continuous speech recognition. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, Minneapolis, MN, Vol. II, pp. 307–310.

Alleva, F., Huang, X. & Hwang, M.-Y. (1996). Improvements on the pronunciation prefix tree search organization. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, Atlanta, GA, pp. 133–136.

Antoniol, G., Brugnara, F., Cettolo, M. & Federico, M. (1995). Language model representations for beam-search decoding. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, Detroit, MI, Vol. 1, pp. 588–591.

Aubert, X. & Ney, H. (1995). Large vocabulary continuous speech recognition using word graphs. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, Detroit, MI, pp. 49–52.

Aubert, X., Dugast, C., Ney, H. & Steinbiss, V. (1994). Large vocabulary continuous speech recognition of *Wall Street Journal* corpus. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, Adelaide, Australia, Vol. II, pp. 129–132.

Baker, J. K. (1975). Stochastic modeling for automatic speech understanding. In *Speech Recognition*, (D. R. Reddy, ed.) pp. 512–542. Academic Press, New York, NY.

Beyerlein, P. & Ullrich, M. (1995). Hamming distance approximation for a fast log-likelihood computation for mixture densities. *Proceedings of the European Conference on Speech Communication and Technology*, Madrid, Spain, pp. 1083–1086.

Bocchieri, E. (1993). Vector quantization for the efficient computation of continuous density likelihoods. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, Minneapolis, MN, Vol. II, pp. 692–695.

Bridle, J. S., Brown, M. D. & Chamberlain, R. M. (1982). An algorithm for connected word recognition. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, Paris, France, pp. 899–902.

Cardin, R., Normandin, Y. & De Mori, R. (1992). High performance connected digit recognition using codebook exponents. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, San Francisco, CA, Vol. I, pp. 505–508.

Dugast, C., Kneser, R., Aubert, X., Ortmanns, S., Beulen, K. & Ney, H. (1995). Continuous speech recognition tests and results for the NAB'94 corpus. *Proceedings of the ARPA Spoken Language Technology Workshop*, Austin, TX, pp. 156–161.

Fissore, L., Giachin, E., Laface, P. & Massafra, P. (1993). Using grammars in forward and backward search. *Proceedings of the European Conference on Speech Communication and Technology*, Berlin, Germany, pp. 1525–1528.

Gauvain, J. L., Lamel, L. F., Adda, G. & Adda-Decker, M. (1994). The LIMSI speech dictation system: evaluation on the ARPA *Wall Street Journal* task. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, Adelaide, Australia, Vol. I, pp. 557–560.

Generet, M., Ney, H. & Wessel, F. (1995). Extensions of absolute discounting for language modeling. *Proceedings of the European Conference on Speech Communication and Technology*, Madrid, Spain, pp. 1245–1248.

Haeb-Umbach, R. & Ney, H. (1994). Improvements in time-synchronous beam search for 10 000-word continuous speech recognition. *IEEE Transactions on Speech and Audio Processing*, **2**, 353–356.

Klatt, D. H. (1980). SCRIBER and LAFS: two new approaches to speech analysis. In *Trends in Speech Recognition* (W. A. Lea, ed.), pp. 529–555. Prentice-Hall, Englewood Cliffs, NJ.

Klovstad, J. W. & Mondshein, L. F. (1975). The CASPERS linguistic analysis system. *IEEE Transactions on Acoustics, Speech and Signal Processing* **23**, 118–123.

Kubala, F., Anastasakos, A., Makhoul, J., Nguyen, L. & Schwartz, R. (1994). Comparative experiments on large vocabulary speech recognition. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, Adelaide, Australia, Vol. I, pp. 561–564.

Kuhn, R. & De Mori, R. (1990). A cache-based natural language model for speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **12**, 570–583.

Lee, C.-H., Giachin, E., Rabiner, L. R., Pieracini, R. & Rosenberg, A. E. (1992). Improved acoustic modeling for large vocabulary continuous speech recognition. *Computer Speech and Language*, **6**, 103–127.

Ljolje, A., Riley, M., Hindle, D. & Pereira, F. (1995). The AT&T 60 000 word speech-to-text system. *Proceedings of the ARPA Spoken Language Systems Technology Workshop*, Austin, TX, pp. 162–165.

Lowerre, B. T. & Reddy, R. (1980). The HARPY speech understanding system. In *Trends in Speech Recognition* (W. A. Lea, ed.), pp. 340–360. Prentice-Hall, Englewood Cliffs, NJ.

Murveit, H., Butzberger, J., Digalakis, V. & Weintraub, M. (1993). Large-vocabulary dictation using SRI's decipher™ speech recognition system: progressive-search techniques. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, Minneapolis, MN, Vol. II, pp. 319–322.

Ney, H. (1982). Connected utterance recognition using dynamic programming. *Fortschritte der Akustik—FASE/DAGA'82, Federation of Acoustic Societies of Europe/Deutsche Arbeitsgemeinschaft fuer Akustik*, Goettingen, Germany, pp. 915–918.

Ney, H. (1993). Search strategies for large-vocabulary continuous-speech recognition. NATO Advanced Studies Institute, Bubion, Spain, June–July 1993. In *Speech Recognition and Coding—New Advances and Trends* (A. J. Rubio Ayuso & J. M. Lopez Soler, eds.), pp. 210–225. Springer, Berlin.

Ney, H., Haeb-Umbach, R., Tran, B.-H. & Oerder, M. (1992*a*). Improvements in beam search for 10 000-word continuous speech recognition. *1992 IEEE International Conference on Acoustics, Speech and Signal Processing*, San Francisco, CA, pp. 13–16.

Ney, H., Mergel, D., Noll, A. & Paeseler, A. (1992*b*). Data driven organization of the dynamic programming beam search for continuous speech recognition. *IEEE Transactions on Signal Processing*, **SP-40**, 272–281.

Odell, J. J., Valtchev, V., Woodland, P. C. & Young, S. J. (1994). A one-pass decoder design for large vocabulary recognition. *Proceedings of the ARPA Spoken Language Technology Workshop*, Plainsboro, NJ, pp. 405–410.

Oerder, M. & Ney, H. (1993). Word graphs: an efficient interface between continuous speech recognition and language understanding. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, Minneapolis, MN, Vol. II, pp. 119–122.

Ortmanns, S. & Ney, H. (1995). Experimental analysis of the search space for 20 000-word speech recognition. *Fourth European Conference on Speech Communication and Technology*, Madrid, Spain, pp. 901–904.

Ortmanns, S., Ney, H. & Eiden, A. (1996*a*). Language-model look-ahead for large vocabulary speech recognition. *Proceedings of the International Conference on Spoken Language Processing*, Philadelphia, PA, pp. 2095–2098.

Ortmanns, S., Ney, H., Seide, F. & Lindam, I. (1996*b*). A comparison of time conditioned and word conditioned search techniques for large vocabulary speech recognition. *Proceedings of the International Conference on Spoken Language Processing*, Philadelphia, PA, pp. 2091–2094.

Renals, S. & Hochberg, M. (1995). Efficient search using posterior phone probability estimates. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, Detroit, MI, Vol. 1, pp. 596–599.

Rosenfeld, R. (1995). The CMU statistical language modeling toolkit and its use in the 1994 ARPA CSR evaluation. *Proceedings of the ARPA Spoken Language Technology Workshop*, Austin, TX, pp. 47–50.
Sakoe, H. (1979). Two-level DP matching—a dynamic programming-based pattern matching algorithm for connected word recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, **ASSP-27**, 588–595.
Schwartz, R. & Austin, S. (1991). A comparison of several approximate algorithms for finding multiple (N-best) sentence hypotheses. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, Toronto, Canada, pp. 701–704.
Soong, F. K. & Huang, E.-F. (1991). A tree-trellis fast search for finding the N-best sentence hypotheses. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, Toronto, Canada, pp. 705–708.
Steinbiss, V. (1991). A search organization for large vocabulary recognition based upon N-best decoding. *Proceedings of the 2nd European Conference on Speech Communication and Technology*, Genova, Italy, Vol. 3, pp. 1217–1220.
Steinbiss, V. (1992). Personal communication. Philips Research Laboratories, Aachen, Germany.
Steinbiss, V., Tran, B.-H. & Ney, H. (1994). Improvements in beam search. *Proceedings of the International Conference on Spoken Language Processing*, Yokohama, Japan, pp. 2143–2146.
Vintsyuk, T. K. (1971). Elementwise recognition of continuous speech composed of words from a specified dictionary. *Cybernetics* **7**, 133–143.
Woodland, P. C., Odell, J. J., Valtech, V. & Young, S. J. (1994). Large vocabulary continuous speech recognition using HTK. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, Adelaide, Australia, Vol. II, pp. 125–128.

(Received 24 July 1996 and accepted for publication 11 November 1996)