A Frame-Synchronous Network Search Algorithm for Connected Word Recognition

CHIN-HUI LEE, MEMBER, IEEE, AND LAWRENCE R. RABINER, FELLOW, IEEE

Abstract—**This paper describes an implementation of a frame-synchronous network search algorithm for recognizing continuous speech as a connected sequence of words according to a specified grammar. Although algorithms of this type have been known since the pioneering work of Vintsyuk, and have been described in a wide variety of implementations, the proposed algorithm implements a couple of new features which were not used in previous approaches. Furthermore, the algorithm, which has all the features of earlier methods, is inherently based on hidden Markov model (HMM) representations, and is described in an easily understood, easily programmable manner (based on pseudocode and detailed block diagram descriptions of the key processing blocks). As such, this paper can be considered a tutorial extension of Ney's paper on dynamic programming algorithms for connected word recognition. The new features of the algorithm include the capability of recording and determining (unique) word sequences corresponding to the best several paths to each grammar node, and the capability of efficiently incorporating a range of word and state duration scoring techniques directly into the forward search of the algorithm, thereby eliminating the need for a postprocessor in the previous implementations. It is also simple and straightforward to incorporate deterministic word transition rules and statistical constraints (probabilities) from a language model into the forward search of the algorithm.**

I. Introduction

The problem of recognizing a fluently spoken sentence (as a string of words or subword units) based on concatenating individual word models (or individual subword models) is extremely important for automatic speech recognition tasks. A wide variety of approaches to this problem, all based on the technique of dynamic programming (DP) [1], have been proposed and evaluated [2]–[13]. The earliest algorithm for connected word recognition was proposed by Vintsyuk [2] who showed how DP techniques could be used to get the optimal sequence of words which match a spoken input. Vintsyuk's procedure processed the speech signal in a frame-synchronous manner, and therefore his pioneering work formed the basis for several DP-based solutions to the speech recognition problems. Vintsyuk also proposed a rudimentary scheme for incorporating syntactic constraints among words in the search (i.e., a grammar).

Since Vintsyuk's work was largely unknown in the U.S. and Japan, several different DP-based search structures were proposed for solving the speech recognition problem, based on concatenation of whole word and subword templates (or models) including the statistical network approach of Baker developed at CMU [3], [4] (which was followed by the research of Lowerre at CMU [5]), the statistical approach by IBM researchers [6], [7], and various template-based word match algorithms [8]–[13]. The key contribution from this early research was the idea of representing all knowledge sources used for recognition (e.g., word representation, language model) as networks (either statistical or deterministic) which could readily be integrated in with the basic network representing speech sounds (e.g., subword or whole word units) and the entire network search could be solved efficiently and accurately using dynamic programming techniques. A variety of the algorithms for finding the "best" path through a network evolved from this research including the stack algorithm developed by Jelinek [14], and a collection of specialized, word-based, dynamic programming search methods such as the two level DP approach of Sakoe [8], the level building algorithm of Myers and Rabiner [9], the one-pass DP approach of Bridle, Brown, and Chamberlain [10], and the one-stage DP approach of Ney [11]. These specialized algorithms were all shown capable of obtaining the best sequence of words which matched a given spoken input string, subject to a wide range of syntactic constraints (word grammar). Differences among the procedures were primarily those of implementation (e.g., frame-synchronous versus word-synchronous), and features (e.g., how duration constraints were imposed, how alternate strings were generated, etc.).

In this paper, which is tutorial in nature to a great extent, we propose a frame-synchronous dynamic programming search algorithm for matching a continuous speech utterance by a connected string of words (or subword units), which retains the best features of all known DP approaches, and which provides the capability of incorporating duration constraints and word transition rules directly into the forward search procedure. We also propose a way to compute the second best path to each grammar node in the grammar network in a frame-by-frame manner so that alternate choices of candidate strings can be obtained easily at the cost of very little extra computation. The algorithm is described in terms of matching sequences of statistical word models (HMM's) although it is equally applicable to subword models and templates with trivial modifications. We use the concept of pseu-

Manuscript received December 28, 1987; revised January 31, 1989.
The authors are with the Speech Research Department, AT&T Bell Laboratories, Murray Hill, NJ 07974.
IEEE Log Number 8930531.

docode and detailed block diagrams to describe both the overall operation of the algorithm, as well as the individual blocks. We show how hardware implementations of the resulting algorithm could be tailored to the specific needs of individual tasks and word vocabularies.

This paper is organized as follows. In Section II, we review the procedure for mapping a given recognition task onto a finite state network (FSN), and show the recognition problem can be formulated as a maximum likelihood path searching optimization problem, which can be solved efficiently, on a frame-by-frame basis, using dynamic programming techniques. In Section III, we review the details of a standard, frame-synchronous procedure for determining the best path through the complete network. In Section IV, we discuss a simple procedure to search for the globally second best path in a given FSN so that alternate choices of candidate strings can be obtained easily even for a very complicated grammar network. In Section V, we present several ways of incorporating statistical word and state duration penalties and word transition penalties directly into the forward computation of the search procedure. In Section VI, we outline the computational complexity of the algorithm, and give ways to achieving reduction in computation for specific tasks. Architectural issues for implementing the algorithm in hardware are also discussed in this section.

II. Speech Recognition Via Finite State Network Decoding

It has been shown by researchers at CMU and IBM that most speech recognition tasks can be organized into a hierarchy of (finite state) networks with a finite number of nodes and arcs corresponding to acoustic, phonetic, and syntactic knowledge sources and their interactions [3]–[7], [14]. Recognition of an utterance corresponds to finding the optimal path through the complete finite state network. This idea of representing all knowledge sources in an FSN has been applied to both isolated word recognition and continuous speech recognition [4], [7], [15]. The optimal network search can be accomplished by sequential decoding using Dynamic Programming (DP) based on a simple concept. The concept was stated by Bellman [1] as *the principle of optimality* in the following terms: "An optimal set of decisions has the property that whatever the first decision is, the remaining decisions must be optimal with respect to the outcome of the first decision." In terms of decoding optimal paths in a finite state network, the principle of optimality enables the decoding to be performed on a frame-by-frame basis, as long as all the information required for the local optimal paths are kept so that the global optimal paths can be found based on the local ones.

For a connected word recognition task, it is instructive to decompose the network into two levels, namely, a phrase (grammar) level and an intraword level as in the formulation of the two-level DP match algorithm [8]. The two levels have completely different properties. The intraword level is usually a word model, which could be a

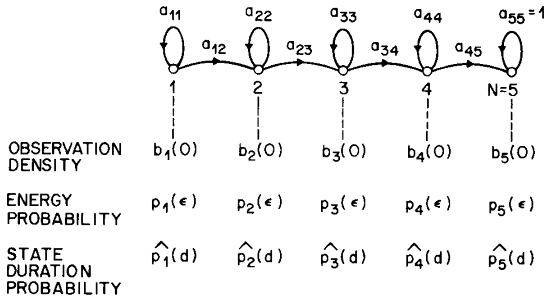

a_{11}
a_{22}
a_{33}
a_{44}
$a_{55}=1$
a_{12}
a_{23}
a_{34}
a_{45}
1
2
3
4
N=5
OBSERVATION DENSITY
$b_1(O)$ $b_2(O)$ $b_3(O)$ $b_4(O)$ $b_5(O)$
ENERGY PROBABILITY
$p_1(\epsilon)$ $p_2(\epsilon)$ $p_3(\epsilon)$ $p_4(\epsilon)$ $p_5(\epsilon)$
STATE DURATION PROBABILITY
$\hat{p}_1(d)$ $\hat{p}_2(d)$ $\hat{p}_3(d)$ $\hat{p}_4(d)$ $\hat{p}_5(d)$

Fig. 1. A 5-state whole word hidden Markov model.

whole word template, a whole word hidden Markov model, or word presentation via an FSN of subword templates or models. In this paper, we will focus our attention on the use of whole word HMM's. An example of a simple 5-state left-to-right HMM, which is also an FSN for representing a word, is shown in Fig. 1. The intraword nodes are essentially the HMM states, while the intraword arcs represent state transitions. For a left-to-right HMM, each intraword node can be reached from only a small number of predecessor intraword states (two in the case of Fig. 1), or from the final state of a valid predecessor model. In general, the intraword level uses a sparse network representation for most recognition tasks.

As for the interword level, it is simply represented by a grammar network (according to the language), in which the nodes represent word boundaries, and the arcs represent whole word models and word transitions. These grammar level network representations range from simple networks with few syntactic constraints to highly constrained, complicated grammar networks. A simple grammar network for a voice dialing application is shown in Fig. 2. The grammar nodes provide points for path merging (e.g., at grammar node 1, the two sets of paths merge), and each grammar arc is labeled with the set of contextually allowable words. By replacing each word on the grammar arcs in Fig. 2 with the corresponding word models (similar to the one shown in Fig. 1), we get a complete finite state network showing all internal and grammar nodes and arcs. From this point on we will only use the simple grammar level network, like the one given in Fig. 2, to represent the complete FSN for a recognition task.

Several examples of the FSN representations of connected digit recognition tasks are given in Fig. 3. In the examples, the grammar nodes labeled "0" are defined as starting nodes, and the grammar nodes with a branching arc labeled "exit" are terminal nodes where a string can terminate. Example 1 [Fig. 3(a)] is a grammar network for recognizing strings with from one to seven digits as is typically used for connected digit recognition problems (e.g., dialing a variable length telephone number). There are seven terminal nodes in the network, and therefore it is possible to generate digit string candidates with from one to seven digits for this task. Example 2 [Fig. 3(b)] is a grammar network that can be used to recognize an ar-

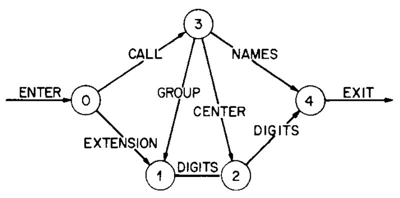


Fig. 2. A grammar network for a simple recognition task.

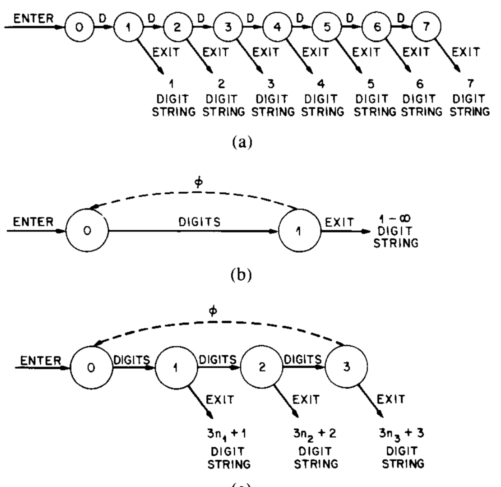


Fig. 3. FSN examples for connected digit recognition. (a) Example 1: a network to recognize up to 7 digits. (b) Example 2: a network to recognize an arbitrary number of digits. (c) Example 3: a network to recognize three possible strings of any number of digits.

bitrary number of digits. The one-pass DP algorithm [10], [11] basically uses this grammar for template-based DTW connected word recognition with unconstrained syntax. The dashed link between the terminal node and the starting node provides a mechanism (a null transition) so that different length partial strings can be merged in the forward search procedure. One problem with this network is that for each spoken utterance, there is only one recognized candidate string, which may or may not be the correct length string. However, compared to the network of Example 1, the number of paths to search is greatly reduced (by a factor of 7). Example 3 [Fig. 3(c)] is an extension of Example 2 which is capable of giving candidate strings of lengths $3n_1 + 1$, $3n_2 + 2$, and $3n_3 + 3$, where n_1, n_2, and n_3 are any nonnegative integers. This FSN can be used to recognize any number of digits and usually the correct length string is included in the set of recognized strings (along with the best string with one insertion and the best string with one deletion). The network of Example 3 is a good compromise between the networks of Examples 1 and 2 in that the search is greatly reduced (from that of Example 2) while candidates of different lengths can still be produced. From our experience with connected digit recognition, the correct string is very likely to be included in the top candidate strings from networks such as that of Fig. 3(c), unless the match between the unknown utterance and the word models is poor and therefore should be rejected. More complicated FSN's include simple pronunciation networks for word models (e.g., [3]-[7]), and more complicated grammar networks, like the Bell Laboratories Airline Reservation System [12].

In order to search for the optimal path in an FSN, some cost measure (e.g., distance, likelihood) associated with a path is required. This cost includes the cost of staying in an intraword internal node, the cost of making transitions from one internal node to another through an intraword arc, the cost of entering a grammar node, and the cost of entering a grammar arc. In a speech recognition task, in which whole word models are characterized by an HMM, the accumulated cost of a path to any node in the FSN at time t can be defined as the negative of the accumulated likelihood of the path at time t, where likelihood is defined as the logarithm of the probability of that path. Therefore, the resulting network is a stochastic FSN where the cost of a path depends on the observation sequence, the duration spent on a certain node, and the transition history of the path.

We now describe how to associate a stochastic cost (penalty) to different parts of an FSN based on using whole word HMM's. For simplicity, we will mix the usage of probability, likelihood, and cost as long as no ambiguity exists. The cost of staying in an internal state at time t is related to the probability of observing the feature vector in that state at time t, and can be defined as the negative of the logarithm of the state observation probability. The cost of making an internal transition includes the negative of the logarithm of the transition probability, plus some possible state duration penalty, which depends on the time spent in the internal state. The cost of entering the right grammar node of a grammar arc includes a possible state and word duration penalty. Finally, the cost of leaving the left grammar node of a grammar arc includes a possible word transition penalty. With all the costs assigned properly, the search for the best path in an FSN is essentially the same as finding the minimum cost path through the network or equivalently performing the maximum likelihood network decoding.

III. A Frame-Synchronous Algorithm for Connected Word Recognition

The DP network search algorithm proposed here is essentially a variation of the frame-synchronous approaches of Vintsyuk, Bridle *et al.*, and Ney [2], [10], [11]. For each node in the FSN, at any time t, the algorithm searches for the best path arriving at that node at that time, and constructs the optimal path of duration t to that node from all the best paths of duration $(t - 1)$. The principle of optimality of DP tells us that the best path to any node i, at time t, can be determined from the best paths to all nodes j, at time $(t - 1)$, plus the best policies (transitions from node j to node i) at time t. Parameters needed for computing the best paths include the accumulated likelihood to any node on the FSN, the path information such as source and destination grammar nodes for each grammar arc on the path, and word and state durations required in order to compute duration probabilities and for traceback. A full description for such a frame-synchronous network search algorithm is as follows.

A Frame-Synchronous Network Search Algorithm:

```
Step 1) Perform initialization for all nodes and traceback buffers
 Step 2) Perform optimal path building frame-by-frame
For every frame of input speech (while not end of utterance)
  Perform feature extraction
  Perform local likelihood computation for all active states in the word models
  For every active grammar node in the network (grammar node loop)
    For every active predecessor grammar node
      For every active grammar arc between them (word model loop)
        Perform local DP for every active internal node (internal state loop)
        Update accumulated path likelihood and path information
      End every active grammar arc
    End every active predecessor grammar node
    Perform path merging DP
    Update accumulated path likelihood and path information
    Update accumulated traceback buffers
  End every active grammar node in the network
End every frame of input speech
Step 3) Perform postprocessing and backtracking
For every active terminal node in the network
  Perform duration scoring postprocessing
  Perform tracebacking to identify recognized string
End every active terminal node
Step 4) Determine the recognized string.
```

It is clear that the frame-synchronous recognition algorithm consists of six main modules, namely: 1) a feature extraction (FE) module to compute the desired feature vector; 2) a local likelihood (LL) module to evaluate the probability of observing the feature vector at all internal states of the word model representations; 3) an intraword decoding (DP1) module to perform a modified Viterbi decoding with duration penalties incorporated in the forward search; 4) a grammar level decoding module (DP2) to perform path merging at every grammar node or level boundary according to the syntactic constraints imposed by the finite-state network; 5) a postprocessing (PP) module to combine duration penalties with path likelihood; and 6) a backtracking (BT) module to decode the optimal candidate strings. The six modules are arranged in the block diagram shown in Fig. 4. Note that DP1, the intraword decoding, can be performed in parallel to the FE and LL modules so that computation can be distributed evenly among several special purpose processors. The "Update" module shown in Fig. 4 uses the local likelihoods computed in the LL module to update the accumulated likelihoods of the optimal paths to all the internal nodes. Modules FE and LL are essentially standard procedure for extracting recognition features and for calculating HMM state likelihoods based on these features. We will now describe modules 3-6 in more detail in the following sections.

A. *A Modified Intraword Viterbi Decoding Algorithm*

Incorporation of Word and State Duration and Word Transition Penalties: Conventional Viterbi decoding algorithms for finding the best decoded sequence for a given hidden Markov model are widely known and described in the literature, e.g., [16]. A duration scoring strategy is generally not incorporated into the algorithm. Even in cases in which state duration penalties were included, they either were of a simplified form, e.g., minimum and maximum durations [5], or were included only in a postprocessor [17]. We will now show how to modify the Viterbi decoding algorithm to include all possible state and word duration probabilities and word transition probabilities so that all the probabilities can be incorporated in the *forward search* scheme either before or after paths are merged at the nodes of the FSN. Fig. 5 illustrates the resulting intraword decoding strategy for a 5-state network ($N = 5$).

The input to the intraword FSN, at time t, is the likelihood score inbest(LG, $t - 1$), of the best path to each grammar node (LG for left grammar node of each word model arc) at time ($t - 1$), as well as the likelihoods, like (i, $t - 1$), of the best path reaching each internal state i at time ($t - 1$). For notational convenience, we also define a likelihood score for state 0 at time ($t - 1$) as

$$\text{like}(0, t-1) = \text{inbest}(\text{LG}, t-1) + \text{word transition penalty}$$

where the word transition penalty is the likelihood of making a transition into the word arc, given the corresponding sequence of words on the best path reaching the left grammar node at time ($t - 1$). The word transition penalty can be deterministic, such that it is zero for all

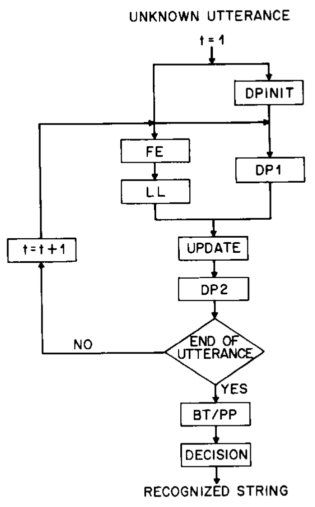


Fig. 4. A block diagram of the frame-synchronous network search algorithm.

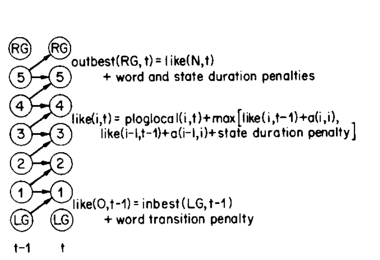

RG
5
4
3
2
1
LG
outbest(RG, t) = like(N,t) + word and state duration penalties
like(i,t) = ploglocal(i,t)+max[like(i,t−1)+a(i,i), like(i−l,t−1)+a(i−l,i)+state duration penalty]
like(0,t−1) = inbest(LG,t−1) + word transition penalty
t−1
t

Fig. 5. An illustration of the modified Viterbi decoding algorithm.

possible transitions and minus infinity for all transitions not allowed in the language of the task. The word transition penalty can also be stochastic, such that it is defined as the logarithm of the probability of a language model (e.g., bigram or trigram). For all intraword nodes, $1 \leq i \leq N$, we have the modified Viterbi maximization such that the likelihood of state i, at time t, like(i, t), is computed as follows:

$$\begin{aligned} \text{like}\,(i, t) = \max\, \big[&\text{like}\,(i, t-1) + \text{a}\,(i, i), \\ &\text{like}\,(i-1, t-1) + \text{a}\,(i-1, i) \\ &+ \text{state duration penalty}\big] + \text{ploglocal}\,(i, t) \end{aligned}$$

where ploglocal(i, t) is the logarithm of the local probability of the feature vector at time t occurring in state i of the word model, $\log(b_i(O_t))$, $a(j, i)$ is the logarithm of the probability of making a transition from state j to state i, and state duration penalty is the logarithm of the probability of having stayed in state $(i-1)$ for d frames, $\log(\hat{p}_{i-1}(d))$, where d is the dwell time in state $(i-1)$. Finally, the likelihood of the best output path reaching the right grammar node RG at time t, called outbest(RG, t) is of the form:

$$\begin{aligned} &\text{outbest}\,(\text{RG}, t) \\ &\quad = \text{like}\,(N, t) + \text{word and state duration penalties} \end{aligned}$$

and the best path from this word arc merges with other paths joining at the same grammar node to produce the best path reaching the grammar node, and it ultimately becomes the input path to other succeeding word arcs in the grammar network, or becomes a path to generate a recognized string at time t if the grammar node happens to be one of the terminal nodes.

The storage arrays needed for performing the modified Viterbi decoding of Fig. 5, including the state dwell time and path elapse time as required for duration scoring, postprocessing, and backtracking, are listed in the following. The time index is dropped from all variables because it is no longer required if a scratch array is added to manage memory storage properly.

like(i)	= accumulated likelihood of the best path to node i
elapse(i)	= duration of the best path to node i since it enters the LG
dwell$(i, 1:i)$	= state dwell times of all states up to node i for the best path to node i
$a(j, i)$	= logarithm of the transition probability from node j to node i
inbest(LG)	= likelihood of the best path entering LG (from previous grammar nodes)
outbest(RG)	= likelihood of the best path reaching RG (into succeeding grammar nodes)
ploglocal(i)	= current observation likelihood for node i
scratch(i)	= temporary scratch array needed to save like(i) at previous frame.

We now describe the modified Viterbi decoding algorithm.

A Modified Viterbi Decoding Algorithm:

```
Step 1: Initialization
        like(0)=inbest(LG) + word transition penalty
        elapse(0)=0
        dwell(0,0)=0
        $a(0,1)=0$
Step 2: compute the best path reaching node $i$, $1 \le i \le N$, at time $t$
        scratch($i$) = like($i$) + a($i$, $i$)
        scratch($i-1$) = like($i-1$) + a($i-1$, $i$) + state duration penalty
        if (scratch($i$) .gt. scratch($i-1$)) then
            elapse($i$) = elapse($i$) + 1
            dwell($i$, $i$) = dwell($i$, $i$) + 1
        else
            scratch($i$) = scratch($i-1$)
            elapse($i$) = elapse($i-1$) + 1
            dwell($i$, $i$) = 1
            dwell($i$, $1:i-1$) = dwell($i-1$, $1:i-1$)
        endif
Step 3: update accumulated state likelihood and produce output path score at time $t$
        like($1:N$) = scratch($1:N$) + ploglocal($1:N$)
        outbest(RG) = like($N$) + word and state duration penalties.
```

As shown in Step 3 above, the local likelihood array is not needed for the DP search. It is only used to incorporate the likelihood of observing the current feature vector when updating the path likelihood.

B. Grammar Node Path Merging Algorithm

Grammar nodes on a network allow paths reaching that node to merge so that only a limited number of the paths are allowed to grow to the succeeding grammar nodes. If only the optimal path is desired at each of the terminal nodes of a grammar network, then path merging at a grammar node eliminates all nonoptimal paths reaching that node. The grammar node path merging algorithm used here is similar to the one used in level building implementation [18] where the optimal path at each grammar node is selected from the set of the output paths from the preceding word arcs reaching that grammar node. For a general grammar network, we not only need a backpointer array, bp, to indicate how long the best path reaching node g has stayed in the best arc, we will also need a node backpointer array, bpnode, to indicate the left grammar node of the best arc reaching that grammar node. An illustration of path merging and path branching at a grammar node is shown in Fig. 6. The storage arrays needed to accomplish the path merging are as follows:

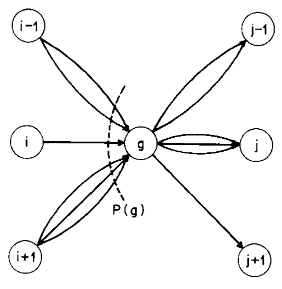
P(g) = the set of all arcs reaching node g = $\{k | \text{RG}(k) = g\}$
glike(g, t) = accumulated likelihood of the best path reaching node g at time t
word(g, t) = identification of the rightmost arc on the best path reaching node g at time t
bp(g, t) = elapse of the rightmost arc on the best path reaching node g at time t
bpnode(g, t) = left grammar node of the best arc reaching node g at time t
outbest(g, k) = likelihood of the best path reaching node g from arc k,

and for every grammar node g, the path merging algorithm can be described in three steps as shown in the following.

```
A Grammar Node Path Merging Algorithm:
Step 1: Perform path merging for all paths reaching
        grammar node g at time t
        $\text{glike}(g, t) = \max_{k \in P(g)} [\text{outbest}(g, k)]$
        $\text{inbest}(g) = \text{glike}(g, t)$
Step 2: Save traceback information of the best path
        reaching grammar node g at time t
        $\text{word}(g, t) = \operatorname{argmax}_{k \in P(g)} [\text{outbest}(g, k)]$
        $\text{bp}(g, t) = \text{elapse}[\text{word}(g, t)]$
        $\text{bpnode}(g, t) = \text{LG}[\text{word}(g, t)]$
Step 3: Use elapse and dwell times to compute duration
        penalties for postprocessing.
```

C. Postprocessing

At the end of the forward search, we have available a list of several possible candidate strings (e.g., string corresponding to best path reaching each terminal node of the grammar network). Postprocessing is a delay-decision strategy to incorporate additional information before the most likely string is selected. The postprocessor used in the level building algorithm [18] incorporates a state duration probability penalty at the end of the level building search so that computation cost is reduced while still achieving comparable performance to Ferguson's internal duration model [19]. In the frame-synchronous search algorithm, it is possible to incorporate either or both of the state and word duration probability penalties in the postprocessor; however, these probabilities can also be accumulated directly in the forward search procedure as shown above, so that postprocessing can be accomplished with very little incremental computational cost.



$$P(g) = \{k \mid RG(k) = g\}$$
$$\text{glike}(g,t) = \max_k [\text{outbest}(g,t,k)]$$
$$\text{word}(g,t) = \operatorname{argmax}_k [\text{outbest}(g,t,k)]$$
$$\text{bp}(g,t) = \text{elapse}[\text{word}(g,t)]$$
$$\text{bpnode}(g,t) = \text{LG}[\text{word}(g,t)]$$

Fig. 6. An illustration of the path merging algorithm at a grammar node.

D. Traceback Procedure

The traceback procedure used to determine the string corresponding to the optimal path through the FSN is basically the same as used in most frame synchronous algorithms and is of the following form.

```
Traceback
  Initialize: $i=g, j=t, k=0$
  While ($j$ .ne. 0)
    $k = k+1$
    $\text{iword}(k) = \text{word}(i,j)$
    $\text{idur}(k) = \text{bp}(i,j)$
    $i = \text{bpnode}(i,j)$
    $j = j - \text{idur}(k)$
  EndWhile
```

After traceback, the number of words in the string is k, and the word sequence of the recognized string and the corresponding word durations can be recovered from the *iword* and *idur* buffers, respectively.

In most applications, we are interested in obtaining a reasonable (although not necessarily optimal) list of candidate strings as soon as the end of the utterance is detected. In this case, traceback information can be saved differently or completely eliminated, while only keeping the accumulated path information to each node in the FSN. For instance, for every desired output string at every terminal node, we can save current string results such as accumulated path likelihood, partial decoded string, and additional accumulated probability to be used in the postprocessor. Therefore, at the end of an utterance, the additional information desired can be incorporated into the accumulated path likelihood and the best string results can be reported right away. However, this results in less flexibility because all the intermediate path information is lost. For example, it is possible that when an end of the utterance is detected, we find that the utterance actually ends earlier; in that case we might prefer that the best strings be determined from the candidate string that terminated at the earlier time. It will also become clear in the next section that if multiple optimal candidate strings are desired at every terminal node of an application network, then the intermediate path information becomes crucial in the search procedure.

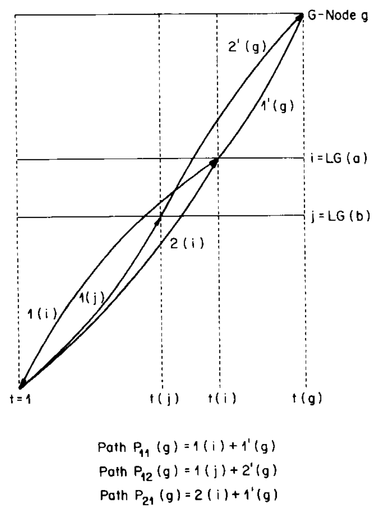

G-Node g
2'(g)
1'(g)
i = LG(a)
j = LG(b)
2(i)
1(i)
1(j)
t=1
t(j)
t(i)
t(g)
Path $P_{11}(g) = 1(i) + 1'(g)$
Path $P_{12}(g) = 1(j) + 2'(g)$
Path $P_{21}(g) = 2(i) + 1'(g)$

Fig. 7. An illustration of the procedure for searching for the second best path.

IV. Search for the Globally Second Best Path to Each Grammar Node

For many connected word recognition applications, it is required that more than one recognition candidate string be presented so that the recognition system can select the best string based on other high level information, such as a database, to verify permissible strings on the candidate list. The system can also present alternative strings to the speaker (in the cases where a recognition error might occur), or decide if alternative strings have sufficiently high scores so that other input is required before making a final recognition decision. Conventional frame synchronous algorithms have not described ways of generating multiple candidate strings at each grammar node. In this section we propose a novel approach to obtain the globally second best paths to all the grammar nodes in an FSN, and the same idea can be extended to include higher rank candidate strings.

By definition, the globally second best path to any grammar node, in a finite state network, is different from the globally best path to that node in the sense that the decoded strings should be different. Since the search is only focused on the grammar node level, rather than on the internal node level, there is very little additional computation required. Similar to the search strategy for the optimal path, searching for the second best path to any grammar node can be accomplished sequentially by computing the second best path to that grammar node at any time t. The second best path information required is also similar to what is needed for the optimal path.

The algorithm proposed here to search for the globally second best path to every grammar node in the finite state network can be formulated based on a concept extended from the principle of optimality. It states that the second best path to any grammar node g, at time t, is either path $P_{12}(g)$ or path $P_{21}(g)$ defined as follows. We define $1(g)$ and $2(g)$ to be the globally best and second best paths reaching grammar node g, and $1'(g)$ and $2'(g)$ to be the locally best and second best subpaths (arcs) on the locally best and second best paths reaching grammar node g. The search idea is illustrated in Fig. 7, where we plot the best path $P_{11}(g)$ and the second best candidate paths $P_{12}(g)$ and $P_{21}(g)$. From the principle of optimality, the best path is composed of the locally best subpath $1'(g) = a$ of duration $\text{bp}(a)$ and the best path $1(i)$ reaching the left grammar node $i = \text{LG}(a)$ of arc a at time $t(i) = t - \text{elapse}(a)$. The candidate path $P_{12}(g)$ is composed of the locally second best subpath $2'(g) = b$ of duration $\text{elapse}(b)$ and the best path $1(j)$ reaching the left grammar node $j = \text{LG}(b)$ of arc b at time $t(j) = t - \text{elapse}(b)$; the candidate path $P_{21}(g)$ is composed of the locally best subpath $1'(g)$ and the second best path $2(i)$ reaching grammar node i at time $t(i)$. Nodes i and j can be the same, and word durations $\text{elapse}(a)$ and $\text{elapse}(b)$ can also be the same; however, the grammar arcs a and b should be different to guarantee that the complete second best path to the grammar node is different from the best path to the same grammar node.

In the algorithm proposed in Section III, we only allow the optimal paths at time $(t - 1)$ to be used in constructing the optimal paths at time t. If we also allow other candidate paths to construct paths in the next level, then the amount of search will increase proportionally to the number of candidate paths retained. It turns out that $P_{12}(g)$ is the second best path among all other paths if paths can only be constructed from the local best ones. Similarly, $P_{21}(g)$ is the second best path if other candidate paths are allowed to grow before they are pruned when merging at the grammar node $\text{LG}(a)$ at time $t - \text{elapse}(a)$. The algorithm proposed here extends the property of the principle of optimality in that only paths $P_{12}(g)$ and $P_{21}(g)$ are compared. As a result, the algorithm requires no additional search computation, and only some additional bookkeeping is required.

The traceback information and duration probabilities of the globally second best paths are managed similarly to that of the globally best paths. However, additional information is required to indicate whether to use the traceback buffers of the best or the second best paths for local traceback so that the correct second best string can be retrieved. We also need a node pointer to indicate where in the network the first branching occurs, so that traceback for the second best path is performed only up to that node.

V. Incorporation of State and Word Duration Probabilities

When intraword state duration is incorporated in an appropriate manner, the performance on some recognition applications improves [20], [21]. These explicit duration models have been shown to significantly increase the computation and storage associated with scoring. A simple alternative, which has been proposed in [18], is to account for the state duration in a postprocessor after the forward level building search is completed. In the following, we focus our discussion on how to incorporate both the word and state duration probabilities into the forward HMM scoring and network search. Details of the exact way in which the word and state durations are measured, and the way in which weights are assigned when these duration penalties are incorporated in the HMM scoring, are given in [18]-[21].

We have shown in Section II how duration penalties can be represented in the network search by associating proper cost to arcs on a finite state network. In Section III-A we also formulated a general strategy showing how the duration penalties can be incorporated in the modified Viterbi decoding algorithm. The way duration penalties are incorporated depends on the HMM scoring strategy in the network search. If they are to be used before the paths merge at a certain node in the network, then they should be added to the path likelihoods before the paths are compared and pruned. If the scoring scheme requires the duration penalties to be included only at the end of the utterance, then a separate buffer is needed to save the additional duration information so that the duration penalties can be imposed by a postprocessor.

The word duration probability is usually included at the end of a word (strategy WW); therefore, candidate paths with unlikely word durations are more likely to get pruned even when the path likelihoods are reasonable. However, we can impose the word duration penalty at the end of an utterance and use a postprocessor to handle the additional information before the best candidate path is selected (strategy WU). We can also use normalized word durations or normalized accumulated word durations in the scoring. Finally, we can consider not using any word duration information at all (strategy W0). The WW strategy was used in the level building search [18] with good success.

In a similar manner, the state duration probability can be included at the end of a state (strategy SS), at the end of a word (strategy SW), or at the end of an utterance (strategy SU) by using a postprocessor. We also can use either normalized state durations within a word or normalized accumulated state durations within a word. We can avoid using the state duration information (strategy S0). The SU strategy was used in [18], again with good success.

The effect of word and duration models on recognition performance often depends on the tasks being evaluated, the feature vectors used, the duration models used, and the amount of training data available to estimate the duration model parameters. In this paper, we are mainly interested in presenting a number of possible word and state duration scoring configurations in a unified manner, so that duration information can be incorporated easily into frame synchronous search algorithms.

VI. Computational Complexity of the Frame Synchronous Algorithm

We have mentioned in Sections III and IV that the grammar level search and the computation of the second best paths only add incremental computational cost. Therefore, most of the computational issues that will be addressed are based on the connected digit recognition grammar shown in Fig. 3(a), where up to seven digits can be recognized. Since the computation wavefront progresses on a frame-by-frame basis, we only need to save path information up to the current frame. The feature extraction module and the local likelihood computation module are basically the same as the ones used in any model-based recognition algorithm, in which the complexity of FE and LL modules depend on the feature vector used and the observation densities selected. Postprocessor and backtracking modules are also similar. We now focus on the storage requirements and the memory management issues for the remaining modules of the algorithm.

A. Computational Considerations

As described in Section III-A, since the modified Viterbi decoding algorithm performs intraword best path search at each frame for all nodes in the grammar, all the best information about the best path has to be saved. The DP search operations involve only additions and comparisons, where additions are used to include transition costs and local likelihoods and to accumulate path likelihoods and optimal path information, while the comparisons are used to perform intraword path merging at the internal nodes. Since the intraword network connection is often sparse, the additions and comparisons generally involve very little computational cost. However, because they lie in the innermost loop of the algorithm, they will have to be performed at every frame, for every model, at every grammar node. For the connected digit grammar shown in Fig. 3(a), if we use 11 words at every grammar arc, use 3 models per word, and 10 states per model to characterize the whole word models, then there are 33 grammar arcs and a total of 330 internal nodes between every connecting pair of grammar nodes. Assuming left-to-right

HMM's with no double jumps are used (e.g., Fig. 1), then there are 2310 comparisons and at least 6930 additions required at every frame. As for the storage requirement, there is no need to save all the computed local likelihoods; only those of the current frame are needed in Step 3 of the algorithm to update the path likelihoods. However, since all path information is required at any time for all nodes, we will need 2310 longwords to save the path likelihood at all internal nodes. We will also need 2310 bytes to store path elapse times, and an optional 23 100 bytes to store the state dwell times for the optimal paths. By way of comparison, for a standard level building algorithm implementation, the storage for one level can be released for higher levels, and therefore local storage is reduced. However, the local likelihoods for all states at all times need to be stored. This requires about 330 longwords per frame, or a total of 99 000 longwords for 300 frames of input speech, which is considerably more storage than required for the frame synchronous implementation.

As for the path merging algorithm described in Section III-B, only comparisons are required to prune nonoptimal paths. For tasks that involve merging many paths at a grammar node, a fast sorting procedure might be required to obtain multiple candidate paths. The memory requirements for the connected digit grammar discussed above are as follows. There are 7 longwords needed at each frame, or a total of 2100 longwords to store the accumulated likelihoods for a 300-frame speech input. In addition, 2100 words are needed to store the word identification array and 2100 words are needed to store the backpointer array used in the traceback. The node backpointer array is not needed for a connected digit grammar, because there is only one fixed predecessor node associated with each grammar node in the network. If postprocessing is required, then an additional array of 2100 longwords are required to save the local duration penalties for each grammar node, at every frame, so that the penalties can be retrieved without recomputing path information in order to use them in postprocessing.

B. Computational Reduction

We mentioned in Section II that, for some applications, the amount of computation can be reduced by using an alternative syntax network. For example, the grammar networks in Fig. 3(b) and (c) can be used to replace the seven digit grammar in Fig. 3(a). Path pruning strategies (e.g., the beam search in [5]) can also be imposed by eliminating paths that fall below some likelihood threshold. The path pruning can be performed on both grammar nodes and arcs. We can also reduce the amount of state dwell time storage by accumulating incremental state duration penalties so that the intraword duration information is no longer needed. However, if state duration penalties are imposed based on normalized intraword state duration proportions, then the dwell time storage cannot be eliminated. Partial traceback [15] can also be used to find optimal partial paths that form the beginning segment of a complete path, so that the search is focused on a smaller subnetwork.

C. Modularized Architecture for Hardware Implementation

The frame-synchronous network search algorithm discussed in Section III has a highly modularized architecture in that most of the computation in the algorithm can be localized so that intermodule communication is minimized. The individual modules have different computational demands so that a special purpose processor can be designed to satisfy the computational needs of each module. For example, the FE and LL modules require a large number of multiplications and additions and therefore require a general purpose digital signal processor. For the DP1 module, a lot of storage is required; however, the search can be accomplished with only additions and comparisons, and therefore a special purpose processor with graph search ability and ample local storage, such as the GSM [13] chip, can be used. For simple applications, module DP2 can also be implemented on general purpose microprocessors or on special purpose processors like the GSM. Finally, modules PP and BT are usually performed on a general purpose microprocessor because the amount of computation is minimal.

The algorithm is also very regular in the sense that the computational requirement is about the same at every time frame. Since the computation is basically repeating a word model loop and a grammar node loop within the frame loop, it can easily be distributed among multiple special purpose, parallel processors. Pipelining the algorithm is also easy, in that different modules can be running in parallel, such that some modules are processing one part of the algorithm and some modules are processing another part. Finally, depending on different application requirements, various modular configurations can be used to implement the task. For example, for speaker-independent, connected digit recognition [18], most of the computation is devoted to evaluating the local likelihoods of a mixture Gaussian density, and more LL modules (relative to DP1 and DP2 modules) are required. Whereas for speaker-dependent, isolated word recognition of a large vocabulary, based on a small number of subword segment models [22], most of the computation will be focused on the intraword search and therefore more DP1 modules than LL modules are needed to implement this task.

VII. Conclusion

In this paper, we have described a frame-synchronous network search algorithm for connected word recognition. The algorithm is based on well-known techniques, but has a couple of features that distinguish it from pr vious implementations, namely: 1) the algorithm .; highly regular, which makes it easy to implement on a general or special purpose hardware; 2) the algorithm is highly modularized so that the local likelihood computation and the local Viterbi decoding for each word model can be performed in parallel, which makes it attractive for distributed computation among multiple special-purpose processors; 3) for decoding a given finite state grammar network, at time t, the algorithm uses only the information about the optimal paths at time $(t - 1)$ plus the observation vector at time t to search for the optimal paths and to update the accumulated path likelihood so that the memory requirement is greatly reduced; and 4) multiple candidate strings, e.g., the word sequence corresponding to the second best path can be obtained easily even for a very complicated grammar network. In addition to the simplified search strategy, the algorithm also has the flexibility that a variety of word and state duration scoring techniques can be implemented simply, efficiently, and directly in the forward search. Word transition rules, such as a language model, can also be easily incorporated in the forward search of the algorithm.

Acknowledgment

The authors would like to acknowledge the work of D. Roe, on a real-time implementation of an algorithm, which is very similar to the one described in this paper. This implementation has been applied to the problem of speaker-independent connected digit recognition, using a special purpose processor and a PC.

References

[1] R. Bellman, *Dynamic Programming.* Princeton, NJ: Princeton University Press, 1957.

[2] T. K. Vintsyuk, "Element-wise recognition of continuous speech composed of words from a specified dictionary," *Kibernetika*, vol. 7, pp. 133-143, Mar.-Apr. 1971.

[3] J. K. Baker, "The dragon system-An overview," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-23, pp. 24-29, Feb. 1975.

[4] ——, "Stochastic modeling for automatic speech understanding," in *Speech Recognition*, D. R. Reddy, Ed. New York: Academic, 1975, pp. 521-542.

[5] B. Lowerre and R. Reddy, "The HARPY speech understanding system," in *Trends in Speech Recognition*, W. Lea, Ed. Englewood Cliffs, NJ: Prentice-Hall, 1980, pp. 340-346.

[6] L. R. Bahl *et al.*, "Automatic recognition of continuously spoken sentences from a finite state grammar," in *Proc. ICASSP 82*, Tulsa, OK, Apr. 1978, pp. 418-421.

[7] L. R. Bahl, F. Jelinek and R. L. Mercer, "A maximum likelihood approach to continuous speech recognition," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-5, pp. 179-190, Mar. 1983.

[8] H. Sakoe, "Two-level DP-matching—A dynamic programming-based pattern matching algorithm for connected word recognition," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-27, pp. 588-595, Dec. 1979.

[9] C. S. Myers and L. R. Rabiner, "A level building dynamic time warping algorithm for connected word recognition," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-29, pp. 284-297, Apr. 1981.

[10] J. S. Bridle, M. D. Brown, and R. M. Chamberlain, "An algorithm for connected word recognition," in *Proc. ICASSP 82*, Paris, May 1982, pp. 899-902.

[11] H. Ney, "The use of a one-stage dynamic programming algorithm for connected word recognition," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-32, pp. 263-271, Apr. 1984.

[12] C. S. Myers and S. E. Levinson, "Speaker-independent connected word recognition using a syntax-directed dynamic programming procedure," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-30, pp. 561-565, Aug. 1982.

[13] S. Glinski *et al.*, "The graph search machine (GSM): A programmable processor for connected word recognition and other applications," in *Proc. ICASSP 87*, Dallas, 1987, pp. 519-522.

[14] F. Jelinek, "A fast sequential decoding algorithm using a stack," *IBM J. Res. Develop.*, vol. 13, pp. 675-685, Nov. 1969.

[15] P. F. Brown, J. C. Spohrer, P. H. Hochschild, and J. K. Baker, "Partial traceback and dynamic programming," in *Proc. ICASSP 82*, Paris, May 1982, pp. 1629-1632.

[16] L. R. Rabiner and B.-H. Juang, "An introduction to hidden Markov models," *IEEE ASSP Mag.*, vol. 3, pp. 4-16, Jan. 1986.

[17] L. R. Rabiner, J. G. Wilpon, and B.-H. Juang, "A performance evaluation of a connected digit recognizer," in *Proc. ICASSP 87*, Dallas, 1987, pp. 101-104.

[18] ——, "A model-based connected-digit recognition system using either hidden Markov models or templates," *Comput., Speech, Language*, vol. 1, no. 2, pp. 167-197, Dec. 1986.

[19] J. D. Ferguson, "Variable duration models for speech," in *Proc. Symp. Appl. Hidden Markov Models to Text and Speech*, J. D. Ferguson, Ed., IDA-CRD, Princeton, NJ, Oct. 1980, pp. 143-179.

[20] S. E. Levinson, "Continuously variable duration hidden Markov models for automatic speech recognition," *Comput., Speech, Language*, vol. 1, no. 1, pp. 29-45, Mar. 1986.

[21] M. J. Russell and A. E. Cook, "Experimental evaluation of duration modeling techniques for automatic speech recognition," in *Proc. ICASSP 87*, Dallas, 1987, pp. 101-104.

[22] C.-H. Lee, F. K. Soong, and B.-H. Juang, "A segment model based approach to speech recognition," in *Proc. ICASSP 88*, New York, Apr. 1988, pp. 501-504.

Chin-Hui Lee (S'79-M'81) was born in July 1951. He received the B.S. degree from National Taiwan University, Taipei, in 1973, the M.S. degree from Yale University, New Haven, CT, in 1977, and the Ph.D. degree from University of Washington, Seattle, in 1981, all in electrical engineering.

In 1981 he joined Verbex Corporation, Bedford, MA, and was involved in research work on connected word recognition. In 1984 he became affiliated with Digital Sound Corporation, Santa Barbara, CA, where he engaged in research work in speech coding, speech recognition, and signal processing for the development of the DSC-2000 Voice Server. Since 1986 he has been with AT&T Bell Laboratories, Murray Hill, NJ. His current research interests include speech modeling, speech recognition, and signal processing.

Lawrence R. Rabiner (S'62-M'67-SM'75-F'75), for a photograph and biography, see p. 1225 of the August 1989 issue of this Transactions.