The Use of a One-Stage Dynamic Programming Algorithm for Connected Word Recognition

HERMANN NEY

Abstract—This paper is of tutorial nature and describes a one-stage dynamic programming algorithm for the problem of connected word recognition. The algorithm to be developed is essentially identical to one presented by Vintsyuk [1] and later by Bridle and Brown [2]; but the notation and the presentation have been clarified. The derivation used for optimally time synchronizing a test pattern, consisting of a sequence of connected words, is straightforward and simple in comparison with other approaches decomposing the pattern matching problem into several levels. The approach presented relies basically on parameterizing the time warping path by a single index and on exploiting certain path constraints both in the word interior and at the word boundaries. The resulting algorithm turns out to be significantly more efficient than those proposed by Sakoe [3] as well as Myers and Rabiner [4], while providing the same accuracy in estimating the best possible matching string. Its most important feature is that the computational expenditure per word is independent of the number of words in the input string. Thus, it is well suited for recognizing comparatively long word sequences and for real-time operation. Furthermore, there is no need to specify the maximum number of words in the input string. The practical implementation of the algorithm is discussed; it requires no heuristic rules and no overhead. The algorithm can be modified to deal with syntactic constraints in terms of a finite state syntax.

I. Introduction

ONE of the most promising approaches to connected word recognition is the technique of dynamic programming [5]. For isolated word recognition, several authors have shown that the problem of nonlinearly time aligning speech patterns with no fixed time scale can be efficiently solved by dynamic programming [6]-[8]. Vintsyuk [1], Bridle and Brown [2], and Sakoe [3] have formulated the connected word recognition problem as an optimization problem similar to the isolated word recognition problem. One of the attractive features of this formulation is the small amount of *a priori* information and training that is required: only the reference patterns for each individual word need to be known. Another advantage is that the three operations of word boundary detection, nonlinear time alignment, and recognition are performed simultaneously; thus, recognition errors due to errors in word boundary detection or to time alignment errors are not possible. The algorithm is forced to match the complete words, and as a result of this, the word boundaries are determined automatically. In comparison, methods based on prespecified segmentation rules [9] or on statistical estimation for segmentation [10] require either detailed knowledge of the vocabulary dependent segmentation rules or an extensive training of the segmentation algorithm. There have been other systems like DRAGON [11] and HARPY [12], which model the recognition problem as an optimization problem as well, but which are based on subword units for the recognition as opposed to whole word templates. Meanwhile, a number of systems for connected word recognition have been developed which are based on the algorithm described in this paper [13]-[16].

Although Vintsyuk had formulated his algorithm already in 1971, his algorithm was not commonly known. Thus, later, Sakoe independently derived a two-level algorithm to solve the optimization problem. On the first level, all reference patterns are systematically matched against all possible subsections of the pattern. This matching operation is the same as for the nonlinear time alignment in isolated word recognition. On the second level, using the distance scores generated, the optimal estimate of the unknown sequence of words is obtained by minimizing the total distance of all possible word sequences. In a recent paper, Myers and Rabiner derived another solution to the optimization problem of connected word recognition [4]. They made use of the property that the matching of all possible word sequences can be performed by successive concatenation of reference patterns. Thus they obtained what they called a level building algorithm.

This paper is essentially tutorial. Its purpose is to present a clear description of the connected word recognition problem and its solution by a one-stage dynamic programming algorithm, to discuss the advantages of the one-stage algorithm and to compare it with other algorithms. The one-stage algorithm is basically identical to the algorithms given by both Vintsyuk [1] and by Bridle and Brown [2]. However, a straightforward formulation and derivation of the algorithm is given. The derivation is based on parameterizing the time warping path by a single index and treating the optimization criterion directly as a function of this time warping path. This approach is much simpler than the approaches used by both Sakoe [3] and by Myers and Rabiner [4], and it results in a computationally more efficient algorithm. The constraints imposed on the path are described in terms of two types of transition rules: transition rules for the word interior and for the word boundaries. Carrying out the optimization using these path constraints and dynamic programming leads to a one-stage algorithm, for which there are no multiple optimization levels as in the other approaches. Unlike the level building of Myers and Rabiner, the one-stage algorithm needs no prespecified maximum number of words in the input string. What is most important, the

Manuscript received February 15, 1982; revised February 7, 1983, July 29, 1983, and October 14, 1983.

The author is with Philips GmbH Forschungslaboratorium Hamburg, D-2000 Hamburg 54, Germany.

one-stage algorithm requires no more computational expenditure than the corresponding case of isolated word recognition with no adjustment window. The implementational aspects of the one-stage algorithm are described, and its computational and storage requirements are compared with the two-level algorithm of Sakoe and the level building algorithm of Myers and Rabiner. Finally, the algorithm is modified to deal with a finite state syntax.

II. Formulation of the Pattern Matching Problem

In the following, we will present a simple approach to the pattern matching problem for connected word recognition, the reason being that the simplification due to parameterizing the time warping path by a single index is most significant and provides some insights that immediately reveal how to arrive at a practical implementation of the algorithm.

Assume an unknown input or test pattern consisting of $i = 1, \cdots, N$ time frames, where a time frame is represented by a vector of features. The input pattern is known to be composed of individual words, which are chosen from a prespecified vocabulary. The words of the vocabulary correspond to a set of K reference patterns or templates obtained from single word utterances spoken in isolation. The word templates are distinguished by the index $k = 1, \cdots, K$. The time frames of the template k are denoted as $j = 1, \cdots, J(k)$, where $J(k)$ is the length of the template k.

The ultimate goal of connected word recognition is to determine that sequence $q(1), \cdots q(R)$ of templates that best matches the input pattern, where the criterion of match needs further specification. The concatenation of the templates $q(1), \cdots, q(R)$ is referred to as "super" reference pattern. Since this unknown "super" reference pattern may be handled like a single utterance pattern, the matching procedure is the same as in the case of isolated word recognition. Based on this consideration, it is obvious what specification and constraints to apply to the time warping procedure. Instead of decomposing the matching procedure into a single template matching level and a word string constructing level, as it was done in the other approaches mentioned [3], [4], we want to treat the matching procedure as a one-stage procedure [1], [2].

The basic idea is illustrated in Fig. 1. The time frames i of the test pattern and the time frames j of each template k define a set of grid points (i, j, k). Each grid point (i, j, k) is associated with a local distance measure $d(i, j, k)$ defining a measure of dissimilarity between the corresponding acoustic events. The connected word recognition problem can be regarded as one of finding the path through the set of grid points (i, j, k) which provides the best match between the test pattern and the unknown sequence of templates. The path is often referred to as time warping path. The three parameters i, j, k are of different characters: the time parameters i and j tend to change more or less uniformly in ascending order, whereas the template number k is constant for comparatively long subsections of the path and can change only after the path has passed a template boundary with $j = J(k)$. However, for deriving the algorithm, it is crucial to treat the three parameters as mathematically equivalent. Formally, the path W is given as a sequence of grid points

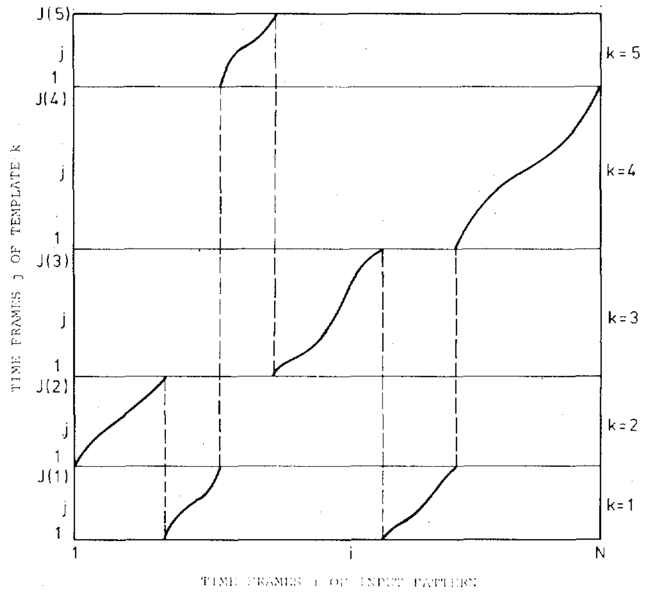

Fig. 1. The connected word recognition problem. The optimal path provides the unknown sequence of words as well as the nonlinear time alignment between the corresponding sequence of templates and the input pattern.

$$W = (w(1), w(2), \cdots, w(l), \cdots, w(L)) \tag{1}$$

where $w(l) = (i(l), j(l), k(l))$ and l is the path parameter for indexing the ordered set of path elements. The criterion for the matching procedure is the global distance, i.e., the sum over the local distances along a given path. The problem of connected word recognition can now be stated as the minimization problem

$$\min_{W} \sum_{l} d(w(l)) \tag{2}$$

i.e., minimize the global distance with respect to all allowed paths. From the best path, the associated sequence of templates can be uniquely recovered as is clear from Fig. 1.

In addition to minimizing the global distance, the time warping path is required to obey certain continuity constraints implied by the physical nature of the patterns to be matched. These constraints apply to consecutive points of the path. The constraints result from the requirement of the preservation of time order along the time axes and from the requirement of time continuity implying that no time frame, i.e., acoustic event, be omitted in the sequence $i(1), \cdots, i(l), \cdots, i(L)$. The continuity constraints determine the possible preceding points for a given path point (i, j, k) and are therefore also referred to as transition rules. A possible disadvantage of the global distance definition as given in (2) is that the global distance depends on the path length, and thus shorter paths are favored. This problem will be studied later in connection with the details of the dynamic programming algorithm.

Due to the concatenation of single word templates to a "super" reference pattern, it is convenient to distinguish between two types of transition rules: transition rules in the template interior called within-template transition rules and

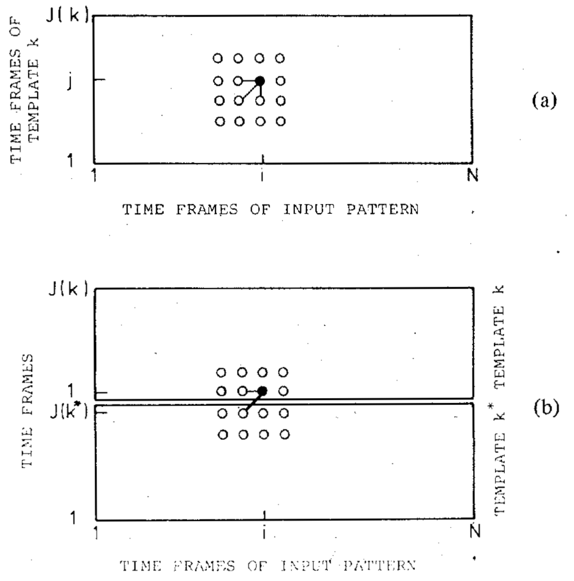
J(k)
j
1
1
i
N
TIME FRAMES OF TEMPLATE k
TIME FRAMES OF INPUT PATTERN
(a)
J(k)
1
J(k*)
1
1
i
N
TIME FRAMES
TEMPLATE k
TEMPLATE k*
TIME FRAMES OF INPUT PATTERN
(b)

Fig. 2. (a) Within-template transition rules. (b) Illustration of between-template transition rules.

transition rules at the template boundaries called between-template transition rules. These two types of transition rules are illustrated in Fig. 2. For within-template transitions the following relation holds between two consecutive points:

if $w(l) = (i, j, k), j > 1$, then

$$w(l-1) \in \{(i-1, j, k), (i-1, j-1, k), (i, j-1, k)\} \qquad (3a)$$

i.e., the point (i, j, k) can be reached only from one of the points $(i-1, j, k)$, $(i-1, j-1, k)$, $(i, j-1, k)$ as shown in Fig. 2(a). For transition at template boundaries where $j = 1$, the between-template transition rules are shown in Fig. 2(b):

if $w(l) = (i, 1, k)$, then

$$w(l-1) \in \{(i-1, 1, k); \; (i-1, J(k^*), k^*): k^* = 1, \cdots, K\}. \qquad (3b)$$

Since the point $(i, 1, k)$ corresponds to the beginning frame of the template k, it is necessary that it can be reached from the ending frame of any template k^* including k itself. The between-template transition rules depend heavily on how the single words can be combined and must therefore be changed in the case of syntactic constraints as will be shown later. The coarticulation problem at the word boundaries is tackled by matching the interior parts of the words such that the word boundaries are correctly treated automatically. Finally, there are endpoint constraints requiring that the time warping path begins at a beginning frame of any template and ends at the ending frame of any template.

III. Derivation of the Algorithm Using Dynamic Programming

In this section, an algorithm for solving the minimization problem (2) and thus finding the best path W is derived by use of dynamic programming. The concept of dynamic programming is especially useful in solving combinatorial optimization problems of the type of (2) which can be broken down into a sequence of optimization steps and in which there are only a small number of possible choices or decisions to be taken at each optimization step [17]. The philosophy of dynamic programming is based primarily on the "principle of optimality" which is due to R. Bellman [5]. Its application to the minimization problem (2) says: If the best path goes through a grid point (i, j, k), then the best path includes, as a portion of it, the best partial path to the grid point (i, j, k).

In order to utilize this "principle of optimality," we define a minimum accumulated distance $D(i, j, k)$ along any path to the grid point (i, j, k). Since the accumulated distance $D(i, j, k)$ is a sum of local distances, it can be decomposed in the same way as the path into the accumulated distance along the best path to its predecessors and the local distance associated with the grid point (i, j, k) itself. To obtain the best path, we have to select the predecessor with the minimum total distance. Thus for the template interior, i.e., $j > 1$, we obtain the following using the within-template transition rules (3a):

$$D(i,j,k) = d(i,j,k) + \min\{D(i-1,j,k), \; D(i-1,j-1,k), D(i,j-1,k)\}. \qquad (4a)$$

At the template boundaries with $j = 1$, the between-template transition rules (3b) yield

$$D(i,1,k) = d(i,1,k) + \min\{D(i-1,1,k); \; D(i-1,J(k^*),k^*): k^* = 1, \cdots, K\}. \qquad (4b)$$

For grid points at the beginning frame of the test pattern, the transition rules must be modified, since there is no preceding frame on the time axis of the test pattern. A grid point $(1, j, k)$ can be reached only from a grid point $(1, j-1, k)$.

Equations (4a) and (4b) are the typical recurrence relations of dynamic programming. By using these recurrence relations, the accumulated distances $D(i, j, k)$ can be recursively evaluated point by point. Evidently, the recursive evaluation is made possible by the transition rules which imply an ordered arrangement of grid points.

By way of summary, a complete algorithm for connected word recognition is given as follows.

```
Step 1) Initialize $D(1,j,k) = \sum_{n=1}^{j} d(1,n,k)$.
Step 2)
  a) For $i = 2, \cdots, N$, do steps 2b-2e.
  b) For $k = 1, \cdots, K$, do steps 2c-2e.
  c) $D(i,1,k) = d(i,1,k) + \min\{D(i-1,1,k); D(i-1,J(k^*),k^*): k^* = 1, \cdots, K\}$.
  d) For $j = 2, \cdots, J(k)$, do step 2e.
  e) $D(i,j,k) = d(i,j,k) + \min\{D(i-1,j,k), D(i-1,j-1,k), D(i,j-1,k)\}$.
Step 3) Trace back the best path from the grid point at a template ending frame with minimum total distance using the array $D(i,j,k)$ of accumulated distances.
```

Step 3 of this algorithm recovers the unknown sequence of words in the input pattern by tracing back the decisions taken by the "minimum" operator at each grid point. For this backtracking procedure, the entire array of accumulated distances $D(i, j, k)$ must have been stored during the recursion. Although this storage requirement of the algorithm is not necessarily intractable even on today's microprocessors, it will be shown in the following section that the storage requirement can be significantly reduced by special techniques.

A drawback of the algorithm formulated so far results from the dependence of the global criterion (2) on the path length. An easy consideration shows that for path segments with a slope greater than 1, the number of local distances per input frame increases, whereas it remains constant for path segments with a slope smaller than 1. Ideally, the global criterion should be independent of the slope of the path in order to allow all types of time axis distortion. At the same time, however, the optimal path is likely to deviate only rarely from the diagonal direction, i.e., to have a slope 1, since speaking rate variations tend to be small. There is no general solution known to this problem. Sakoe and Chiba [8] describe two within-word transition rules for which the path slopes are always $\frac{1}{2}$, 1, or 2, and the path length is equal to the number of test frames processed. The Itakura constraints [7] lead to the same path length normalization, i.e., no time distortion penalties for slopes between $\frac{1}{2}$ and 2. Locally variable penalties for time distortion can be introduced as illustrated in Fig. 3. Depending on the three directions horizontal, diagonal and vertical, the local distance is multiplied by the weights $(1 + a)$, 1, and b prior to evaluating the dynamic programming recursion:

$$\begin{aligned} D(i,j,k) = \min \{&(1 + a) \cdot d(i,j,k) + D(i-1,j,k), \\ &d(i,j,k) + D(i-1,j-1,k), \\ &b \cdot d(i,j-1,k) + D(i,j-1,k)\}. \end{aligned}$$

The reformulation of the global criterion (2) is omitted since it is trivial.

Fig. 3 indicates also what is the number of local distances per input frame for slopes 2, 1, and $\frac{1}{2}$. Typical values of the weights a and b are in the order of 1, e.g., $a = 1$ and $b = \frac{1}{2}$.

IV. Practical Implementation of the Algorithm

The final aim of the algorithm is to determine the unknown sequence of words. In order to accomplish this, it is sufficient to know at which time frame i of the test pattern the best path has started for a given ending point of a given template k.

The details of the best path within the templates are of no primary importance to the recognition problem. Another important aspect for reducing the storage requirement is evident from the structure of loops in the algorithm presented in the previous section. To perform the dynamic programming recursions for a time frame i, only a small portion from the complete array $D(i, j, k)$ of accumulated distances is needed, namely the elements corresponding to the preceding time frame i: $\{D(i-1, j, k)$: $k = 1, \cdots, K; j = 1, \cdots, J(k)\}$. The grid points associated with these elements form a vertical cut through the time plane of Fig. 1. This column of storage will be simply referred to as column array of accumulated distances and denoted as $D(j, k)$. Thus, using only one column of storage from the array $D(i, j, k)$, the dynamic programming recursions (3a) and (3b) can be carried out by proceeding along

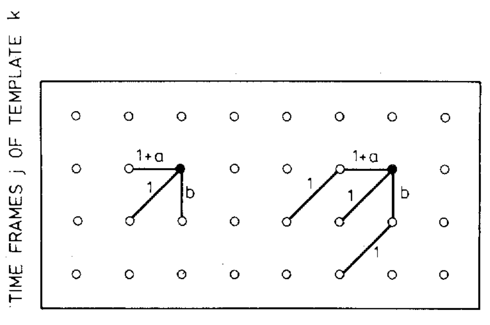

Fig. 3. Time distortion penalties as provided by slope dependent weights $1 + a$, 1, b: The number of local distances per input frame is thus $1 + (a/2)$ for $\frac{1}{2}$, 1 for slope 1 and $1 + b$ for slope 2.

the time axis of the test pattern and updating the storage column point by point.

This technique for storage reduction is analogous to the case of isolated word recognition, where all that is wanted is the minimum total distance as a matching score between the test pattern and the template. The essential difference, however, is that some form of backtracking must be added to enable the algorithm to recover the unknown sequence of words. Bridle *et al.* [13] refer to the following technique for backtracking as Vintsyuk's algorithm [1]. A similar backtracking procedure was proposed by Myers and Rabiner [4]. The backtracking information must be recorded during the evaluation of the dynamic programming recursion. For the path leading to the grid point (i, j, k), there is a unique starting point at the line $j = 1$ in the same template k. Hence, for each grid point, a backpointer $B(i, j, k)$ can be defined as that value of the ending frame of the preceding word from which the path to the grid point (i, j, k) has come. Fig. 4 illustrates the basic concept of the backpointers for the three preceding grid points of the grid point (i, j, k). Originally, the array of backpointers depends on the index triple (i, j, k) in the same way as the array $D(i, j, k)$ of accumulated distances.

Since, as stated above, we are only interested in the backpointers of the grid points $(i, J(k), k)$ at the template boundaries, we can reduce the array of backpointers $B(i, j, k)$ to a column array of backpointers $B(j, k)$ (in Fig. 1) as in the case of the array $D(i, j, k)$, and update it point by point according to where the best path has come from. A "from template" array $T(i)$ must be used to record the index k_0 of the template with minimum accumulated distance at its ending frame $J(k_0)$ for each frame i of the test pattern. Formally, this is expressed as

$$\begin{aligned} T(i) = k_0 = \operatorname{argmin} \{&D(i, J(k), k) \\ &\hat{=} D(J(k), k): k = 1, \cdots, K\}, \end{aligned}$$

where the operator "argmin $\{f(x)$: $x = x_0, \cdots, x_n\}$" means to find the optimum argument x which minimizes $f(x)$. Additionally, a "from frame" array $F(i)$ must be introduced in order to retain the backpointer $B(J(k_0), k_0) \hat{=} B(i, J(k_0), k_0)$ to the ending frame of the preceding word after the test pattern frame i has been processed. Thus the "from frame" array $F(i)$ keeps track of the frame along the test pattern time axis from which the best path to the grid point $(i, J(k_0), k_0)$ has come. In other words, the "from frame" array keeps track of

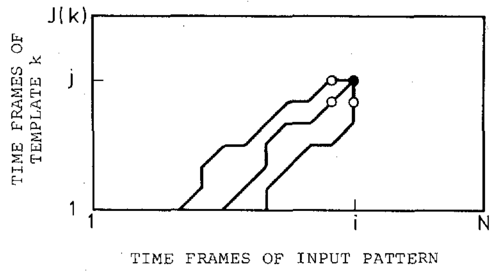

Fig. 4. Backpointers from the three preceding grid points (i, j, k) to their corresponding starting frames.

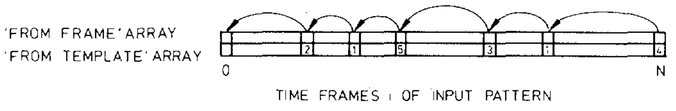

Fig. 5. The backtracking procedure.

potential word boundaries, and the "from template" array keeps track of the respective decision about the recognized word.

The concept of the "from template" and of the "from frame" arrays is closely related to the between-template transition rules, which are the characteristic of this approach presented as opposed to the other approaches. It is crucial to realize that for each time frame i, only the best template, i.e., the template with minimum accumulated distance at its ending frame, and the corresponding word boundary must be kept track of in order to be able to determine the optimal global path.

Fig. 5 shows an example of the backtracking procedure for recovering the unknown sequence of words for the example in Fig. 1. For the last time frame of the test pattern, the "from template" array records the template with minimum total distance at its ending frame. The "from frame" array points to the ending frame of the preceding word on the test pattern time axis. For this ending point, the "from template" array again contains the index of the best preceding word and the corresponding "from frame" array points again back to the ending frame of the preceding word. Obviously, the "from frame" array contains indices into itself and the "from template" array. This backtracking is continued until the zeroth frame of the test pattern has been reached. Thus for the optimal word boundaries we obtain the sequence (in reverse order)

$$N, F(N), F(F(N)), \cdots,$$

and for the optimal word sequence, we obtain the sequence (in reverse order)

$$T(N), T(F(N)), T(F(F(N))), \cdots.$$

In summary, the array $D(i, j, k)$ of accumulated distances along with the associated array $B(i, j, k)$ of backpointers has been substituted by four smaller arrays. These arrays are as follows.

The column array of accumulated distances

$$D(j, k) := D(i, j, k),$$

the column array of backpointers

$$B(j, k) := B(i, j, k),$$

the "from template" array

$$T(i) := \operatorname{argmin} \{D(i, J(k), k): k = 1, \cdots, K\}$$

the "from frame" array

$$F(i) := B(J(T(i)), T(i)) \mathrel{\hat{=}} B(i, J(T(i)), T(i)).$$

The savings achieved by using these arrays instead of the original array $D(i, j, k)$ are considerable: instead of $N \cdot \bar{J} \cdot K$, where $\bar{J}$ denotes the average length of a template, only $2 \cdot (\bar{J} \cdot K + N)$ storage locations are required.

Fig. 6 shows a schematic diagram of the complete algorithm using the arrays introduced above and provides a short verbal description of the one-stage dynamic programming algorithm for connected word recognition.

In order to allow for the possibility of pauses between the words in the word string, it is useful to include an additional silence template in the set of templates. This technique can also be used for refining the detection of the beginning and ending points of the complete word string as it was proposed by Bridle *et al.* [13].

These authors also suggested an extension of the algorithm to deal with nonvocabulary words. They introduced a so-called pseudotemplate consisting of only one frame to which a fixed "distance" is assigned independent of with which input frame it is matched. Thus, the algorithm is able to reject a portion of the input utterance if the similarity is not sufficient.

The connected word recognition algorithm described so far starts the traceback from the end of the word string. However, it is possible to recognize the initial portion of the word string before its end has been spoken, which is very desirable for a real time operation of the recognition system. This is clear from the observation that if there is a section shared by all candidate paths, that section is necessarily part of the globally optimal path [13], [14]. Thus, by tracing back all candidate paths for a given input frame and determining the point where they all meet, the initial portion of the word string can be recognized. This recognition is irrevocable in the sense that no subsequent input can change the decision about the recognized words. In order to carry out this premature traceback, it is in fact sufficient that decisions about the recognized words as such, and not about the path details, are the same for all candidate paths.

V. Syntactic Constraints

In this section, the algorithm is modified in order to include a syntactic analysis for a finite state syntax. For connected word recognition, it is suitable to incorporate the syntactic analysis into the recognition process itself. It is worth noting that any lexicon of legal word strings can be described in terms of a finite state syntax. Clearly, syntactic constraints must affect the between-word transition rules. Each word as it starts can be reached only from words that may legally precede it according to the syntactic constraints. Thus, we define for each word k a "from predecessor" set $P(k)$ as the set of all vocabulary words which can precede the word k. Fig. 7(a) illustrates the method. For convenience, START and STOP are added as pseudowords to the vocabulary $\{A, B, C\}$. The arcs of the finite state network are associated with the words of the vocabulary. The finite state network shown in Fig. 7(a) is

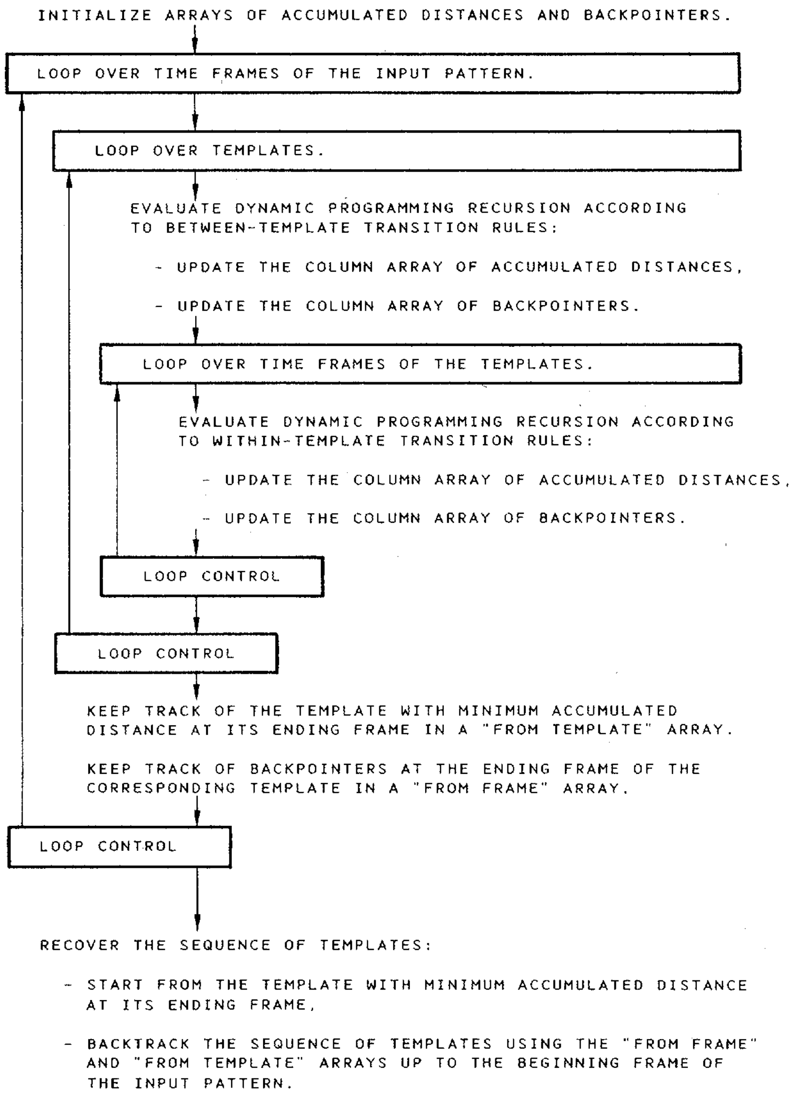
INITIALIZE ARRAYS OF ACCUMULATED DISTANCES AND BACKPOINTERS.
LOOP OVER TIME FRAMES OF THE INPUT PATTERN.
LOOP OVER TEMPLATES.
EVALUATE DYNAMIC PROGRAMMING RECURSION ACCORDING TO BETWEEN-TEMPLATE TRANSITION RULES:
- UPDATE THE COLUMN ARRAY OF ACCUMULATED DISTANCES,
- UPDATE THE COLUMN ARRAY OF BACKPOINTERS.
LOOP OVER TIME FRAMES OF THE TEMPLATES.
EVALUATE DYNAMIC PROGRAMMING RECURSION ACCORDING TO WITHIN-TEMPLATE TRANSITION RULES:
- UPDATE THE COLUMN ARRAY OF ACCUMULATED DISTANCES,
- UPDATE THE COLUMN ARRAY OF BACKPOINTERS.
LOOP CONTROL
LOOP CONTROL
KEEP TRACK OF THE TEMPLATE WITH MINIMUM ACCUMULATED DISTANCE AT ITS ENDING FRAME IN A "FROM TEMPLATE" ARRAY.
KEEP TRACK OF BACKPOINTERS AT THE ENDING FRAME OF THE CORRESPONDING TEMPLATE IN A "FROM FRAME" ARRAY.
LOOP CONTROL
RECOVER THE SEQUENCE OF TEMPLATES:
- START FROM THE TEMPLATE WITH MINIMUM ACCUMULATED DISTANCE AT ITS ENDING FRAME,
- BACKTRACK THE SEQUENCE OF TEMPLATES USING THE "FROM FRAME" AND "FROM TEMPLATE" ARRAYS UP TO THE BEGINNING FRAME OF THE INPUT PATTERN.

Fig. 6. Schematic diagram of the one-stage dynamic programming algorithm.

uniquely specified in terms of the "from predecessor" sets:

$$P(A) = \{\text{START}\},$$
$$P(B) = \{A, B\},$$
$$P(C) = \{\text{START}, A, B\},$$
$$P(\text{STOP}) = \{C\}.$$

In general, however, the "from predecessor" sets may depend on the position or context of the word in the finite state network. An example is shown in Fig. 7(b) where the word B requires two different "from predecessor" sets depending on its position. To deal with this problem of position dependence, we use the following method: we make several copies of each word, such that the finite state network is uniquely described by the "from predecessor" sets. For the example in Fig. 7(b), we have to make two copies B_1 and B_2 of the word B as shown in Fig. 7(c). Thus, we obtain the four-word vocabulary $\{A, B_1, B_2, C\}$ and the "from predecessor" sets:

$$P(A) = P(B_1) = \{\text{START}\},$$
$$P(B_2) = P(C) = \{A, B_2\},$$
$$P(\text{STOP}) = \{B_1, C\}.$$

This method of making copies can also be used for prescribing the number of words in the word strings as illustrated in Fig. 7(d). The legal word strings described by the network of Fig. 7(d) consist of three words from the vocabulary $\{A, B\}$.

The recognition algorithm can treat the copies of a word as if they were different words. Given the "from predecessor" sets, the syntactic constraints are then captured by the following modification of the recurrence relation for the between-word transitions:

$$D(i, 1, k) = d(i, 1, k) + \min \{D(i-1, 1, k);$$
$$D(i-1, J(k^*), k^*): k^* \in P(k)\}$$

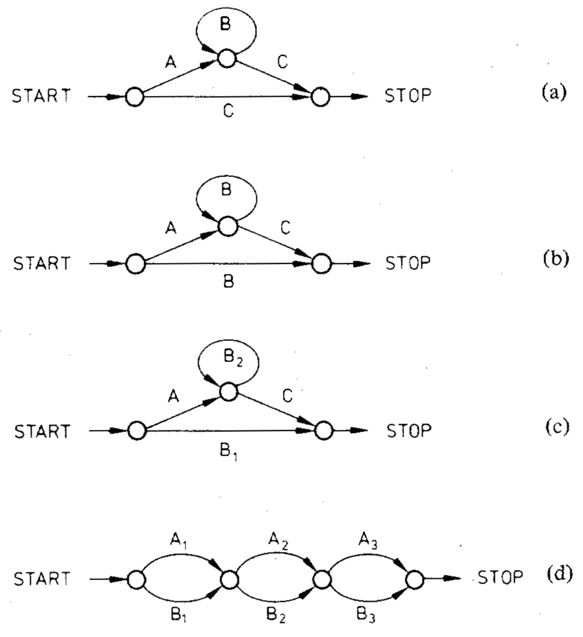

Fig. 7. Four examples (a), (b), (c), (d) of finite state networks as syntactic constraints for building up word strings from the vocabulary words A, B, C.

where the index k now indexes the word copies instead of the word classes. The recurrence relation for the within-word transitions is not changed. Note that the finite state networks need not be loop free, which was already indicated by Fig. 7.

Since now the optimal preceding word depends on the word under consideration, it is necessary to have a "from template" array $T(i)$ and a "from frame" array $F(i)$ for each word copy. For the several copies of a word, the local distances need to be calculated only once for one copy and can then be "copied" for the other copies. As a result, there is no significant increase in computational expenditure due to the syntactic constraints. The major part of the computation time is consumed in the most inner loop by the calculation of the local distance $d(i, j, k)$, which is a distance measure between feature vectors of dimension ten and more. Hence, the additional costs for the computation and updating of the backpointers, which is carried out in parallel with the computation of the accumulated distances, are negligible in comparison with the local distance calculations. There is, however, an increase in storage costs since each word copy must have its individual arrays for the accumulated distances and the backpointers.

VI. Comparison With Other Approaches

It is instructive to compare the different approaches to connected word recognition with respect to computational and storage requirements. Myers and Rabiner [4] present such a comparison between the two-level algorithm of Sakoe [3], their original level building algorithm and a reduced level building algorithm, which results from the level building algorithm by restricting the search region for the best path. In the following, we will extend their comparison to include the one-stage dynamic programming algorithm. Since, as we have seen, syntactic constraints do not significantly affect the computational requirements of the one-stage algorithm, the comparison is made for word strings with no syntactic constraints.

The computational and storage requirements of the four algorithms are summarized in Table I. The computational requirements are measured as the number of local distance calculations, which is the product of basic time warps multiplied by the (average) size of a time warp, i.e., the area covered by it. According to Myers and Rabiner [4], the storage requirements include only the storage locations for those arrays that are specific to the different algorithms. Thus the storage locations for the reference patterns as well as the arrays of accumulated distances and backpointers are not considered in general. However, as will be seen later, there will be an exception for the one-stage algorithm.

In the two-level algorithm of Sakoe, a time warp is carried out for each of the N time frames of the input pattern and for each of the K templates with average length $\bar{J}$. Each time warp covers a stripe around the diagonal line of bandwidth $(2R + 1)$; the warp size is then $\bar{J} \cdot (2R + 1)$. As a result, the number of local distance calculations is $N \cdot K \cdot \bar{J} \cdot (2R + 1)$. To start up the matching procedure on the second level, the socalled phrase level, the accumulated distance and the index of the best template must be stored for each of the $(2R + 1)$ pairs of beginning and ending points and for each time frame of the input pattern. Thus, the number of storage locations is $2 \cdot N \cdot (2R + 1)$.

In the level building algorithm of Myers and Rabiner, $M \cdot K$ time warps are performed, where M is a prespecified number for the maximum number of words, i.e., levels, in the input string. Each time warp covers an area of $\bar{J} \cdot N/3$ grid points. The factor $\frac{1}{3}$ is due to the Itakura constraints imposed on the local slope of the time warping path. These constraints result in a global restriction of the search area for the path. Hence, the total number of distance calculations is $M \cdot K \cdot \bar{J} \cdot N/3$. At each of the M levels and for each of the N time frames of the input pattern, the accumulated distance, the best template and the starting position on the input pattern time axis must be kept track of. This requires $3 \cdot N \cdot M$ storage locations. The level building algorithm turns out to be equivalent to the one-stage algorithm with a prescribed maximum number of words in the string: the levels correspond to the copies made of each template as described in the preceding section. For the reduced level algorithm, the size of a time warp is $\bar{J} \cdot (2R + 1)$, where R is a range parameter and defines a reduced search region for the path. It is worthwhile noting, however, that the reduced level algorithm is not guaranteed to find the globally optimal sequence of words.

The one-stage algorithm performs one time warp for each template. Strictly speaking, the term time warp is not appropriate for this algorithm, since actually only one global time warp is performed. However, the area covered by this global time warp as shown in Fig. 1 can be thought of as decomposed into smaller time warps for each template, as far as local distance computations are concerned. Each of these smaller time warps is of size $\bar{J} \cdot N$. Hence, the total number of local distance computations is $K \cdot \bar{J} \cdot N$. Since the one-stage algorithm must

TABLE I
COMPUTATIONAL COMPARISONS OF DYNAMIC PROGRAMMING ALGORITHMS FOR CONNECTED WORD RECOGNITION AND TYPICAL COMPUTATIONAL REQUIREMENTS FOR VOICE DIALING (i.e., 12 DIGITS IN A STRING)

	Two-Level Algorithm	Level Building Algorithm	Reduced Level Building Algorithm	One-Stage Algorithm
Number of basic time warps	$K \cdot N$	$K \cdot M$	$K \cdot M$	K
Size of time warps	$\bar{J} \cdot (2R + 1)$	$\bar{J} \cdot N/3$	$\bar{J} \cdot (2R + 1)$	$\bar{J} \cdot N$
Total computation	$K \cdot N \cdot \bar{J} \cdot (2R + 1)$	$K \cdot M \cdot \bar{J} \cdot N/3$	$K \cdot M \cdot \bar{J} \cdot (2R + 1)$	$K \cdot \bar{J} \cdot N$
Storage	$2 \cdot N \cdot (2R + 1)$	$3 \cdot N \cdot M$	$3 \cdot N \cdot M$	$2 \cdot (N + K \cdot \bar{J})$
Number of basic time warps	3600	120	120	10
Size of time warps	875	4200	875	12600
Total computation	3150000	504000	105000	126000
Storage	18000	12960	12960	1420

where $\bar{J} = 35$ = average length of a template
$K = 10$ = number of templates
$M = 12$ = maximum number of words in the input string, e.g., for voice dialing (5 + 7) digits
$N = 360$ = length of the input string
$R = 12$ = range parameter for time warping

keep track of the accumulated distances and backpointers for all vocabulary words at a time, it is appropriate to include the accumulated distances and the backpointers in the storage costs. Thus, the storage locations required are $2 \cdot (\bar{J} \cdot K + N)$ as shown in Section IV. The above considerations show the one-stage algorithm to be the only algorithm that calculates each local distance exactly once. Furthermore, the amount of computation and storage required for the one-stage algorithm is independent of the maximum allowed number of words in the input string as opposed to the level building algorithm and its reduced version.

Computational reductions by pruning techniques are also applicable to the one-stage algorithm [12]-[14]. They are based on the observation that candidate paths with accumulated distances significantly larger than the minimum accumulated distance for the input frame under consideration are very unlikely to result in the optimal path. Candidate paths with accumulated distances exceeding a threshold set relative to the best path candidate for the input frame considered can be removed from the subsequent operations.

Hence, the pruning operation requires the following additional computational steps to be performed for each grid point in connection with the evaluation of the dynamic programming recursion: one comparison for determining the minimum of the accumulated distances and three comparisons concerning the accumulated distances of the predecessor grid points for deciding whether to prune the grid point under consideration or not. As a result, the pruning overhead involves four comparisons per grid point, which definitely require less than a few percent of the overall computation time per grid point without pruning. For the pruned grid point or the corresponding path candidate, no local distance is calculated. The pruning technique results in an adjustment window of varying size [13]: when there is a very well defined minimum for the accumulated distances, only a few path candidates are retained; on the other hand, when there is no clear minimum at all, all or nearly all path candidates are expanded. A conservative estimate of the reduction factor achieved by pruning is three or more; apart from a few extra storage locations, there is no increase in storage requirements.

Table I gives also a numerical comparison of the computational and storage requirements of the four algorithms for some typical parameter values:

$$M = 12, \ K = 10, \ N = 360, \ \bar{J} = 35, \ R = 12.$$

Such parameter values are appropriate for the application of voice dialing, where there are, e.g., twelve digits in the input string: five digits for the area code and seven digits for the telephone number. For such long digit strings, it may be useful to include the endpoint detection in the recognition algorithm as described in Section V. Apart from the parameters M and N, the values are the same as chosen by Myers and Rabiner [4]. Table I shows for this example that the one-stage algorithm requires only $\frac{1}{25}$ of the computational amount of the two-level algorithm of Sakoe and only $\frac{1}{4}$ of the computational amount of the level building algorithm of Myers and Rabiner. As for the storage requirements, the one-stage algorithm offers a reduction factor of 9 or more as compared to the three other algorithms. Only the reduced level building algorithm does with a computational amount comparable with the one-stage algorithm, however, without ensuring the global optimality of the solution. Moreover, by using pruning the computational expenditure of the one-stage algorithm could be reduced further.

VII. Summary

In this paper, a tutorial presentation of a one-stage dynamic programming algorithm for connected word recognition and its relation to other dynamic programming algorithms has been given. The one-stage algorithm is basically the same as the algorithms given by Vintsyuk [1] and developed by Bridle and Brown [2]. The derivation of the one-stage algorithm is based on parameterizing the time warping path by one single index and results in a one-stage dynamic programming algorithm. The dynamic programming strategy performs three functions simultaneously: the time alignment, the word boundary detection and the classification itself. Thus, the problem of concatenating reference patterns of a connected word string is solved in a highly efficient manner. The main features of the one-stage algorithm include the following.

1) For input strings with no syntactic constraints, the algorithm is independent of a prespecified maximum number of words in the input string.

2) The computational effort is proportional to the number of time frames in the reference patterns and in the input string; each local distance between a reference frame and an input frame is calculated exactly once. In particular, the computational effort per input frame turns out to be exactly the same as in the case that the input string consists of isolated words and an isolated word recognition with no adjustment window is performed for each word.

3) The algorithm lends itself to a straightforward and simple implementation.

4) Syntactic constraints, formulated in terms of finite state syntaxes, can be dealt with by simple modifications of the algorithm and increase the computational expenditure only negligibly.

Acknowledgment

The author thanks the reviewers for helpful and constructive comments.

References

[1] 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.

[2] J. S. Bridle and M. D. Brown, "Connected word recognition using whole word templates," in *Proc. Inst. Acoust. Autumn Conf.*, Nov. 1979, pp. 25-28.

[3] 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.

[4] 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.

[5] R. Bellman, *Dynamic Programming.* Princeton, NJ: Princeton Univ. Press, 1957.

[6] T. K. Vintsyuk, "Speech discrimination by dynamic programming," *Kibernetika*, vol. 4, pp. 81-88, Jan.-Feb. 1968.

[7] F. Itakura, "Minimum prediction residual principle applied to speech recognition," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-23, pp. 67-72, Feb. 1975.

[8] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-26, pp. 43-49, Feb. 1978.

[9] M. R. Sambur and L. R. Rabiner, "A statistical decision approach to the recognition of connected digits," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-24, pp. 550-558, Dec. 1976.

[10] R. Zelinski and F. Class, "A segmentation procedure for connected word recognition based on estimation principles," in *Proc. 1981 IEEE Int. Conf. Acoust., Speech, Signal Processing*, Atlanta, GA, pp. 960-963, Mar.-Apr. 1981.

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

[12] B. T. Lowerre, "The HARPY speech recognition system," Ph.D. dissertation, Carnegie Mellon Univ., Dep. Comp. Sci., Pittsburgh, PA, Apr. 1976.

[13] J. S. Bridle, M. D. Brown, and R. M. Chamberlain, "An algorithm for connected word recognition," in *Proc. 1982 IEEE Conf. Acoust., Speech, Signal Processing*, Paris, France, May 1982, pp. 899-902.

[14] P. F. Brown, J. C. Spohrer, P. H. Hochschild, and J. K. Baker, "Partial traceback and dynamic programming," in *Proc. 1982 IEEE Conf. Acoust., Speech, Signal Processing*, Paris, France, May 1982, pp. 1629-1632.

[15] J. Peckham, J. Green, J. Canning, and P. Stephens, "Logos–A real-time hardware continuous speech recognition system," in *Proc. 1982 IEEE Conf. Acoust., Speech, Signal Processing*, Paris, France, May 1982, pp. 863-866.

[16] H. Ney, "Connected utterance recognition using dynamic programming," in *Proc. 3rd Cong. FASE, DAGA*, Goettingen, Germany, Sept. 1982, pp. 915-918.

[17] —, "Dynamic programming as a technique for pattern recognition," in *Proc. 6th Int. Conf. Pattern Recogn.*, Munich, Germany, Oct. 1982, pp. 1119-1125.

Hermann Ney was born in Saarlouis, Germany, in 1952. He received the Diplom degree in physics from Goettingen University, Goettingen, Germany, in 1977 and the Dr.-Ing. degree in electrical engineering from Braunschweig Technical University, Braunschweig, Germany, in 1982.

Since 1977, he has been with Philips Research Laboratories, Hamburg, where he has worked on speaker verification via telephone lines, digital signal processing, word recognition, and speech recognition. His work has concentrated on the application of dynamic programming to the problem of decision making in context, such as nonlinear time alignment, nonlinear smoothing, and pitch contour determination. He is particularly interested in the application of mathematical techniques in signal processing research and in pattern recognition research. He is currently responsible for research on continuous speech recognition.