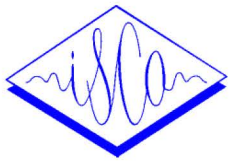

First European Conference on
Speech Communication and Technology
EUROSPEECH '89
Paris, France, September 27-29, 1989

GENERATION OF MULTIPLE HYPOTHESIS IN CONNECTED PHONETIC-UNIT RECOGNITION BY A MODIFIED ONE-STAGE DYNAMIC PROGRAMMING ALGORITHM

José B. Mariño, Enrique Monte

Dpto. Teoría de la Señal y Comunicaciones. UPC.Spain

ABSTRACT

One of the most popular algorithms for connected word (or subword phonetic unit) recognition is the one-stage dynamic programming algorithm. In its available formulation, this algorithm is not designed to provide multiple hypothesis; such a limitation is currently becoming a drawback, since the need of alternative recognitions in the systems presently under research is being acknowledged. This paper introduces a modified version of the one-stage dynamic programming algorithm tailored to afford multiple hypothesis.

I.- INTRODUCTION

Most of continuous speech recognition systems being currently under research realize separately the acoustic and linguistic analysis of utterances to be recognized. In other words, the acoustic and linguistic processing are done at two different stages of the recognition task; the output provided by the acoustic processor is the input material to the linguistic processor, which completes the recognition operation. In order to facilitate the task of the linguistic subsystem, it is usual that the acoustic analyzer yields multiple hypothesis; this is performed via a lattice of recognition phonetic units (syllables, demisyllables, diphones, etc.) obtained by spotting.

In systems using phonetic units like syllables, demisyllables and similar ones, the acoustic analysis can be implemented by tools arising from connected word recognition systems. Some examples of connected demisyllables (or syllable) based systems can be given for large vocabulary word recognition and also for continuous speech recognition . In order to make evolve these systems to the above mentioned more powerful schemes it is necessary to provide additional capacities to the connected phonetic-unit recognition algorithm. Perhaps, the most important of these capacities is the ability of multiple hypothesis generation.

A typical algorithm applied to connected word (phonetic-unit) recognition is the so-called one-stage dynamic programming algorithm [1]. After reviewing this algorithm, this paper introduces a modified version designed to afford n different strings of phonetic units that match the utterance (or a part of it) with the n best scores (n being a parameter).

II.- ONE-STAGE DYNAMIC PROGRAMMING ALGORITHM

The one-stage dynamic programming algorithm is well known: an excellent description of it can be found in reference [1]. In this section a brief summary is provided in order to review its basic principles and introduce the nomenclature used along this paper.

Let us consider a test pattern represented by i = 1, . . ., N time frames, where each frame is characterized by a vector of features. This test is supposed to be a realization of a sentence appertaining to a given language, for which the recognition system has been trained to recognize.

Every sentence in this language is represented by a sequence of phonetic units; thus, the language can be defined by a set of phonetic units and a grammar that specifies the correct sentences of the language as sequences of phonetic units. A convenient structure for this grammar is a finite state o regular grammar [1], [2], where every state represents a phonetic unit and the links indicate the allowed transitions. Hence, the system recognizes a test pattern as a sequence of phonetic units that correspond to an allowed sequence of states.

In general, a phonetic unit has to be represented by more than one state in order to cope with the different contexts where the unit can appear; as a consequence, we may introduce the concept of syntactic unit (state of grammar) as generalization of a phonetic unit. Accordingly we may say that a test pattern will be recognized as a sequence of syntactic units.

If every phonetic unit is represented by a conveniently trained template, we can associate this template to the corresponding syntactic units of the grammar. Let S(k), k = 1, . . . , K be the template of state k; the finite state grammar imposes that this template can succeed only a determined subset of templates, which is denoted as P_k (the subset being constituted by the templates corresponding to the predecessor states of state k [2]). There are to special subsets of particular interest; the subset P_o, that includes the syntactic units that a allowed sequence can start with, and the subset P_∞, that indicates the syntactic units that can be allocated at the final end of a sequence. The time frames of a template k are denoted as j=1,..., J(k), where J(k) is the length of the template k.

The goal of the one-stage dynamic programming algorithm is to determine the sequence of templates that best matches the test pattern, while verifying the syntactic constraints. If we collect all the templates in order to built a "super template", we can look at the one-stage algorithm as a modified dynamic time warping algorithm, that determines the optimum path relating the time frames of the test pattern to the frames in the "super template". A point in this path is defined by three indexes (i,k,j); thus, the local distance between the frame i of the test and the frame j of the template k is denoted as d(i,k,j); the distance accumulated by the optimum path at the point (i,k,j) is denoted as D(i,k,j).

Optimum path means that path with the minimum accumulated distance. This path is determined by a sequential way, starting with frame i=1 and finishing in frame i=N. The construction of this path has to obey some continuity constraints, i.e., the point (i,k,j) may be reached only from a predefined set of points w(i,k,j). The point (i,k,j) presents two situations of particular relevance here :

a) the point is within a template k:

$$1 < j \leq J(k)$$

in this case w(i,k,j) has to define a typical dynamic time warping production set; for instance, the Itakura's production set

$$w(i,k,j) = \{(i-1,k,j),\ (i-1,k,j-1),(i-1,k,j-2)\}$$

b) the point corresponds to the beginning (j=1) of a template k; this point is allowed to be reached from the end of the predecessor templates of k; thus

$$w(i,k,1) = \{(i-1,k,1);(i-1,k',J(k')) \ :\ k' \ \varepsilon\ P_k\}$$

In the sequel, these transition rules will be called within templates and between templates, respectively.

Although a path is defined by the set of points by which the path runs over, as far as the recognition proposes are concerned we are interested only in the corresponding sequence of templates. As a consequence, it is enough for the recognition goal to keep the information about the transitions between templates. Accordingly, two data have to be conserved : the template k_back(i,k) from which the transitions to the template k took place in the frame i, and the frame i_back (i,k) where the path started in the template k_back(i,k). In order to be able to set up these data, a temporary information appears to be necessary: the frame back(k,j) where the optimum path passing by (i,k,j) has started in the template k.

The one-stage dynamic programming algorithm can be formulated with the following steps:

Step 1 : Initialization

if $k \varepsilon P_o$:

$D(1,k,1) = d(1,k,1)$

back (k,1) = 1

if $k \notin P_o$ or $j \neq 1$:

$D(1,k,j) = \infty$

back (k,1) = -1

For i =2 to N

For k =1 to K

Step 2 : transition between templates

$$D(i,k,1) = d(i,k,1) + \min_{w(i,k,j)} D(i', k', j')$$

If the point of w(i,k,j) minimizing D(i', k', j') corresponds to the end of the template k* (i.e. a transition between templates has occurred), it is necessary to define

k_back(i,k) = k*

i_back(i,k) = back(k*,J(k*))

Step 3 : transition within templates

$$D(i,k,j) = d(i,k,j) + \min_{w(i,k,j)} D(i',k,j')$$

The point (i,k,j) is reached from the point (i*,k,j*) of w(i,k,j) that presents the minimum accumulated distance :

back(k,j) = back(k,j*)

End i

End k

Step 4 : Optimization

The optimum path accumulates the minimum distance at the final frame of a template allowed to be at the end of a sequence. Let k* be the last template of the

optimum sequence; it verifies :

$$D(N,k^*,J(k^*)) < D(N,k,J(k)) \qquad k,\ k^* \in P_\infty$$

This template k^* begins at the frame

$$i^* = back\ (k^*,\ J(k^*))$$

Step 5 : Backtracking

The optimization step identifies both the final template k^* of the optimum sequence and the frame i^* where this template k^* started. From k_back (i^*,k^*) and i_back (i,k^*) we set the predecessor template of k^* and its beginning. Thus, by an iterative procedure, we can recover the overall sequence of templates corresponding to the optimum path.

III.- GENERATION OF MULTIPLE HYPOTHESIS

Now, our interest is addressed to obtain the n sequences of syntactic units that best match the test pattern. In general, the one-stage algorithm in its actual formulation is unable to yield this result. Since it is oriented to provide only the optimum sequence, the transition rules do not take into account but the best hypothesis, discarding the rest of possible paths to the point (i,k,j). In order to be sure that we can get the n best accumulated distances, we have to follow the n best paths along the space (i,k,j) ; in other words, we have to consider the n best transitions anywhere. This idea is almost unfeasible; fortunately, as we are interested in generating the n best sequence of syntactic units rather than the n best paths in the space (i,k,j), we can formulate a simplified implementation of the n-path generalization of the one-stage dynamic programming algorithm. In this reduced version, only the transition between templates is generalized to n alternatives; the rest of the algorithm decisions are made under the best path hypothesis, i.e. the optimum path drives the paths within a template and determines the frame i where a transition between templates will take place.

Thus, a new algorithm able to provide multiple hypothesis can be formulated by implementing minor modifications of the one stage dynamic programming algorithm. In the sequel, this new algorithm is described by paralleling the previous description of the one-stage algorithm. However, since we will define n paths at a point (i,k,j), we need to generalize some terms used before by adding a new argument m that indicates the path. So, the distance will be denoted by D(i,k,j,m) and similarly we introduce k_back (i,k,m) and i_back (i,k,m).

A formulation of the new algorithm follows :

Step 1 : Initialization

For m = 1 to n

if $k \in P_0$:

$D(1,k,1,m) = d(1,k,1)$

back (k,1) = 1

if $k \notin P_0$ or $j \neq 1$:

$D(1,k,j,m) = \infty$

back (k,1) = -1

end m

For i=2 to N

For k=1 to K

Step 2 : Transition between templates

$$D(i,k,1,1) = d(i,k,1) + \min_{w(i,k,1)} D(i',k',j',1)$$

If the point in w(i,k,j) minimizing D(i', k',j',1) is not (i-1,k,1), a transition between templates occurs. In such a case, the algorithm examines the set of accumulated distances

$\{D(i-1,\ k',J(k'),m):\quad k' \in P_k\ ,\ m=1,\ldots,n'\}$

looking for the n smallest distances (n' will be defined below).

Let k(m) and D(i-1,k(m), J(k(m)),m') be the template that provides the m-th distance and the corresponding m-th distance, respectively (in general $m' \leq m$); the algorithm sets the accumulated distances after the transition as follows :

$D(i,k,1,m) = d(i,k,1) + D(i-1,k(m),J(k(m)),m')$

and actualizes the backtracking pointers :

k_back (i,k,m) = k(m)

i_back (i,k,m) = back (k(m), J(k(m)))

Step 3 : Transition within templates

$D(i,k,j,m) = d(i,k,j) + D(i^*,k,j^*,m)$

where the point $(i^*,\ k,\ j^*)$ of w(i,k,j) minimizes D(i',k,j',1). The back (k,j) is set to back (k,j^*).

End k

End i

In Step 2 we have to distinguish to situations :

a) A transition between the first and the second syntactic units of the sequence; in this case,

$back(k',J(k')) = 0$

and the algorithm takes n'=1, because

$$D(i,k',J(k'),m) = D(i,k',J(k'),1) \qquad m = 1, \ldots, n$$

b) Otherwise, the algorithm takes n'=n.

Step4 : Optimization

At frame N of the test pattern, the n minimum accumulated distances of the set

$$\{D(N,k,\ J(k),\ m);\quad k \in P_\infty\ ,\ m=1, \ldots, n\}$$

are determined. In this way, we obtain the last template k*(m) corresponding to the m-th best sequence and identify in which path m*(m) of k(m) the m-th distance was detected. The backtracking information is preserved by defining

$$i^*(m) = back\ (k^*(m),\ J(k^*(m)))$$

Step 5 : Backtracking

This operation follows a similar way as that of the one-stage algorithm. From k*(m), i*(m) and m*(m) we get the predecessor template of k*(m) in the m-th sequence as

$$k^{**}(m) = k\_back\ (i^*(m),\ k^*(m),\ m^*(m))$$

and its beginning as

$$i^{**}(m) = i\_back\ (i^*(m),\ k^*(m),\ m^*(m))$$

In order to be able to continue the backtracking process, we have to be able to determine the m' path of k**(m) that generated the m*(m) path in k*(m). Obviously this information could have been kept as k_back and i_back were; however, it is not necessary, because m' can be obtained at every transition between templates by the following expression

$$m' = \sum_{p=1}^{m} \delta\ (k(m),\ k(p))$$

where

$$\delta(s,t) = \begin{cases} 1 & \text{if } s=t \\ 0 & \text{otherwise} \end{cases}$$

Thus, the backtracking can continue to the starting template of the m-th sequence.

IV.- CONCLUSION

In this paper a modification of the one-stage dynamic programming algorithm for connected words has been introduced; the new algorithm is able to provide the most likely sequences of phonetic units corresponding to a test pattern. This algorithm has been tested in our demisyllable based system for number recognition [3]; in every trial, the issued sequences matched the sentences expected according to the phonetic similarities; this result was considered as a practical validation of the new algorithm.

The main drawback of the algorithm, is the increasing of computational burden as the number of hypothesis to be generated grows. One way to reduce the computational burden is to realize that in most of the cases the p best hypothesis can be obtained by preserving only n < p paths in every transition between templates. Other way to lighten this problem, surely more effective, passes by changing the exhaustive search of the algorithm by a convenient heuristic search. When the phonetic units are represented by several templates, a computational burden reduction is definitely necessary.

V.- REFERENCES

[1] H. Ney , "The use of an one-stage dynamic programming algorithm for connected word recognition", IEEE Trans. ASSP-32, pp 263-271 : April, 1984.

[2] J. B. Mariño et al., "Finite state grammar inference for connected word recognition", Proc. EUSIPCO'88, pp 1035-1038 : Sept., 1988.

[3] J. B. Mariño et al., "Recognition of numbers and strings of numbers by using demisyllables: one speaker experiment", this issue.