CMU-CS-85-181

Incremental Network Generation in Template-Based Word Recognition

Kai-Fu Lee

Department of Computer Science
Carnegie-Mellon University
Pittsburgh, PA 15213

December 1985

Abstract

The network representation of word templates is presented. Using the network method, words are divided into segments, and same or different word templates can share segments. This not only reduces storage required, but also enables the system to focus on acoustically dissimilar segments. Yet, by retaining multiple examples of the same word, it encapsulates variations of speech. A word recognition system has been designed and implemented to facilitate network training by providing (1) relatively reliable segmentation, (2) a segment-based warping algorithm that tolerates inexact segmentation, and (3) incremental network generation. Two network generation methods for isolated word recognition are presented, one of which achieved 99% accuracy for speaker-dependent alphabets, 92% for speaker-independent alphabets, and 99% for speaker-independent digits. Several other methods of reference generation were implemented using the same system, and the incremental network technique proved to be superior to all of them.

This research was sponsored by a National Science Foundation Graduate Fellowship, and by Defense Advanced Research Projects Agency Contract N00039-85-C-0163. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the National Science Foundation, the Defense Advanced Research Projects Agency, or the US Government.

Table of Contents

List of Figures

List of Tables

1. Introduction

In order for any template-based speech recognition system to perform well, it must have a well-chosen reference set against which input speech will be matched. While it is possible to select and modify templates manually (such as in HARPY [1]), a great amount of expertise and time is required. Consequently, many procedures for automatic reference set generation were developed.

A reference generation procedure takes a number of repetitions of each word in the vocabulary, and produces a reference set consisting of one or more templates per word. A procedure generating exactly one template per word is a *single-template training method.* A procedure that could generate several templates per word is a *multiple-template training method.*

The simplest form of reference generation is *Single-Template Casual Training* [2]. Using it, a set of the training data is installed as the reference. It is the worst of all the methods because it is seriously impeded by unreliable templates. An improvement over *Casual Training* is *Reference Selection* [3], which considers multiple instances of each word, and selects the token that is most likely to lead to correct recognition. However, this technique requires a large amount of computation, and the improvement is limited.

It is not necessary to use natural templates as in *Single-Template Casual Training* and *Reference Selection.* One simple method is *Averaging Training* [4], in which all replications of the same word are averaged to create a single template for that word. Another method, *Simplified Clustering* [2], selectively finds two templates for each word to be averaged as the reference template.

All of the above methods yield a single template for each word. For the speaker-dependent case, this is usually adequate because within-speaker differences tend to be relatively small. For speaker-independent recognition, however, one single template cannot capture the between-speaker differences [5].

A simple but inelegant solution is to take multiple replications of each word, and use *all* of them as the reference set. This is the *Multiple-Template Casual Training.* While this produces fine results, there is an enormous amount of duplication since many replications of the same word are similar. Treating each replication as a distinct reference template is very inefficient, since recognition of an input requires space and time proportional to the number of reference templates.

Statistical Clustering [6] is a method that compares favorably with the results of *Multiple-Template Casual Training,* yet greatly reduces the space and time required. A clustering algorithm takes a large number of replications of the same word, and identifies clusters of similar tokens. Rather than using every token as in

Multiple-Template Casual Training, the tokens in a cluster are collapsed into one template (typically by averaging). Since the tokens in a cluster are similar to each other, there is no loss of information. Consequently, clustering eliminates much of the within-word duplications.

This method, however, does not eliminate all the duplication of acoustic information. For example, the letters B and D share the common vowel /i/. Clustering algorithms are incapable of generalizing between-word similarities by collapsing the two /i/ segments into one. This type of generalization is perhaps more important than the within-word generalization because after averaging the /i/ segments, the two templates share the same /i/ part. Now, any variations in the /i/ of an input B or D is irrelevant because the match score for that segment will be the same against the two templates. Therefore an input would match B better than D if and only if its fricative portion matches /b/ better than /d/. Furthermore, this will reduce time and storage requirements.

It is with these goals in mind that we investigate the *Network Representation* of speech. Although these theoretically desirable properties are easy to visualize, in reality, the generation of the network is a non-trivial task. Two incremental network generation methods for isolated word recognition are presented in this study. The first method, the *learning* approach, treats training data as if they were input data. The system modifies the network for each word in the training data depending on whether the word was correctly recognized. The second method, the *matching* approach, tries to match every segment against every other, and combine similar ones. The *matching* approach produced results superior to all of the abovementioned systems in both speaker-dependent and speaker-independent recognition.

There are two problems inherent in network training: (1) The network method seems to crucially depend on almost perfect segmentation, which is impossible with current knowledge and technology, and (2) standard dynamic programming (DP) of whole word templates is difficult to apply because we must align particular segments together so that the advantages of the network representation will materialize. Yet, DP alignments often do not completely agree with segmentation boundaries.

Our solution is a word recognition system that uses:

- Segment-based recognition over the entire utterance.
- Modified Zapdash [7] segmentation.
- A hybrid warping procedure that tolerates inexact segmentation.
- Branch and bound exhaustive search.
- The cepstrum distance metric with power and duration information.

As shown below, this system is accurate as well as quite efficient.

We begin this paper by presenting the word recognition system. Next, the two network generation approaches are described and discussed in detail. Finally, we compare the results of network training with each of the other techniques discussed above.

2. The Word Recognition System

The word recognition system shown in Figure 2-1 can be used with any type of reference generation. To implement a new reference generation technique, it is only necessary to substitute the "Reference Generation" block. The same signal processing and segmentation are applied to training as well as input data.

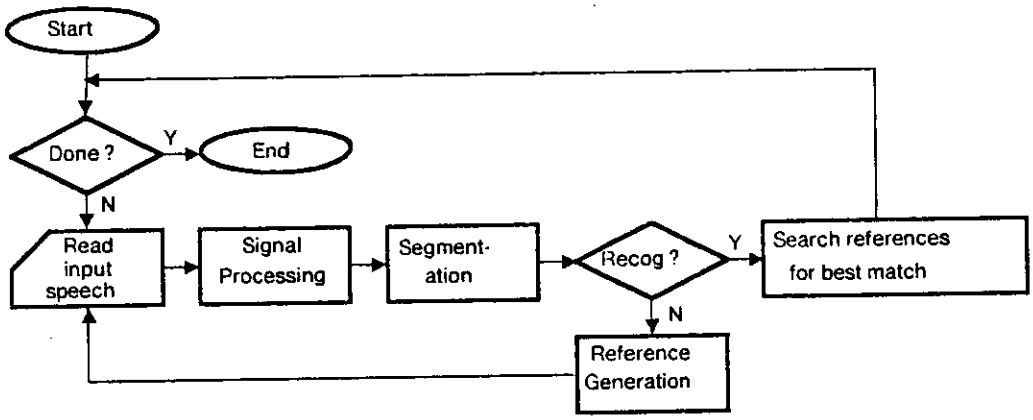

Figure 2-1: The word recognition system.

2.1 Database

Three existing isolated word databases were used:

1. **Speaker Dependent Alphabets** - 15 tokens of each letter of the alphabet of one speaker were used. 5 of the tokens were used to generate the reference set. The remaining 10 served as test data.
2. **Speaker Independent Alphabets** - 5 male and 5 female speakers each produced 4 tokens of each letter of the alphabet. To generate the reference set, 2 tokens of each speaker were used (20 sets), and the remaining 2 tokens of each speaker (20 sets) were used as test data.[1].
3. **Speaker Independent Digits** - 5 male and 5 female speakers each produced 4 tokens of each of the 10 digits (zero to nine). To generate the reference set, 2 tokens of each speaker were used (20 sets), and the remaining 2 tokens of each speaker (20 sets) were used as input data.

The only exception to the above conventions is *Single-Template Casual Training*, which uses only one set of templates as its reference set. In that case, we used each of the 5 (or 20 for speaker-independent) sets, and the results shown are the average of the 5 (or 20) experiments. Other than this exception, the above conventions are strictly followed in all systems to facilitate comparisons among them.

[1]Note that some literature refers to this as *multiple speaker* and not *speaker independent* because the same speakers are used to train and test the system.

2.2 Signal Processing and Representation of Speech

The database was recorded using a high quality microphone in an office-like environment. The speech is sampled at 10 KHz. Each speech waveform is multiplied by a hamming window, and 12 autocorrelation coefficients are computed using a window size of 24 milliseconds, incremented every 3 milliseconds. Each 24 millisecond window is thus represented as a *frame* of speech data. These autocorrelation coefficients are preemphasized using a simple one zero filter with a preemphasis factor of 0.7. LPC analysis is then performed, producing 24 LPC cepstrum coefficients. The power value for each frame is calculated and normalized by the maximum value in the utterance. Both the cepstrum coefficients and power are compressed into a 9-bit representation.

2.3 Segmentation

Although there are other network systems [1] [8] [9], none was totally segment-based[2]. This is largely due to inaccuracies in segmentation. The network system presented here is segment-based, and we try to cope with imperfect segmentation by using (1) broad category segment labels, (2) a fast and relatively accurate segmentation algorithm, and (3) a warping algorithm that does not crucially depend on perfect segmentation. In this section we discuss (1) and (2). (3) will be discussed in 2.4.2.

The segmenter used is a subset of the Zapdash segmenter [7]. The standard Zapdash segmenter works in 6 cycles that provide successively finer segments. For the purpose of this study, it was halted after 3 cycles, which provided sufficient information to generate the desired labels: **Vocalic** (voiced), **Fricative** (unvoiced), and **Silence**. Experiments showed that more precise labels lead to more segmentation errors, which negatively affect recognition. The Zapdash output is post-processed to eliminate unrealistically short segments. Figure 2-2 shows an example of Zapdash-segmentation of the letter "X". The number under the segment label indicates the duration in milliseconds.

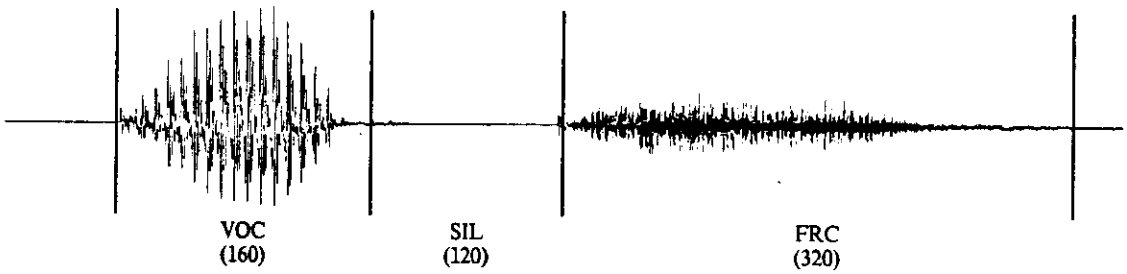

Figure 2-2: Zapdash segmentation of the letter X.

[2] In this study, every input utterance (training or test) is segmented, and recognition, as well as matching, depends upon segmentation. In most other network systems, however, segmentation is only applied to the training data.

Although Zapdash sometimes misses weak fricatives, and occasionally detects noise as actual segments, its performance is satisfactory for the purpose of this study. In order to show its adequacy, 15 sets of the alphabet data were hand segmented into vocalic, fricative, and silence regions. A frame by frame comparison of the hand segmentation and the Zapdash segmentation showed different labels in only 3.56% of the 10 millisecond frames. Using *Single-Template Casual Training* (See Section 5.1) for those 15 sets of one speaker, the recognition accuracies obtained were 91.98% and 91.59% for the hand segmentation and Zapdash segmentation respectively. This shows that Zapdash segmentation for coarse class labels is quite reasonable.

2.4 Recognition

The recognition module will be examined in a bottom up fashion: first, how the frame to frame distance is computed; second, how two words are aligned frame to frame; finally, how the best matching reference word for the input is found.

2.4.1 The Cepstrum Distance Metric with Duration and Power

The distance metric used in the system is a modified cepstrum distance that considers not only the spectral difference, but also power and duration differences. Many studies have dealt with the cepstrum distance [10] [11] [12]. The metric presented here is adopted from Shikano's study [12], with some variations. Standard cepstrum distance between two frames is defined by:

$$\sum_{k=1}^{N}(C_{ik}-c_{jk})^2$$

where C_{ik} and c_{jk} represent the k^{th} cepstrum autocorrelation coefficient of the i^{th} frame of the reference and the j^{th} frame of the test data. $N(=24)$ is the order of LPC analysis.

An enhancement takes power into consideration. Power is computed for every frame of the data, and combined into the equation as follows:

$$W_p \times (P_i-p_j)^2+\sum_{k=1}^{N}(C_{ik}-c_{jk})^2$$

where i and j are the indices of the frame being compared, and P_i and p_j are the power values for the reference and input frames. W_p is a weight applied to the power so that it does not become too prominent. Our experiments have shown that optimal performance can be achieved when the average contribution of power (to the total distance) is about 6-7% in the speaker dependent recognition, and 11-12% in speaker independent recognition.

Another improvement is to consider the durational difference between the reference and the input. After the frame-wise distances have been computed and summed, we add to that the durational difference

between the two segmental groups [See next section] multiplied by some constant, expanding the above equation to:

$$W_d|D-d| + \sum_{i=1}^{D} \{ W_p \times (P_i - p_j)^2 + \sum_{k=1}^{N} (C_{ik} - c_{jk})^2 \}$$

where W_d is the weight for duration, D is the number of frames in this group of the reference, and d is the number of frames in this group of the input. Experiments show that optimal performance can be achieved when the average contribution of duration is 3% in speaker dependent recognition, and 4% in speaker independent recognition.

To show the contribution of power and duration, three *Single-Template Casual Training* (See Section 5.1) experiments were run for both the speaker-dependent and speaker-independent alphabet databases using distance measurements obtained from simple cepstrum distance, cepstrum distance with power, and cepstrum distance with power and duration. Table 2-1 shows the results. It is evident that adding power and duration has improved the recognition, particularly for speaker-independent data. This is because spectral information is less reliable between different speakers.

Distance Measurement	Speaker Dependent	Speaker Independent
Cepstrum Distance	90.00%	47.69%
Cep. Dist. + power	90.76%	50.76%
Cep. Dist + power + duration	91.53%	52.30%

Figure 2-1: Alphabet recognition results using single template training with three variations of cepstrum distance.

2.4.2 Hybrid Warping Algorithm

Warping is the alignment of the input and the reference (mapping each frame in the reference to a corresponding frame in the input, so that they can be compared). The most simplistic kind of warping, linear warping, simply takes the two samples and matches them according to the ratio of their lengths. An example of linear warping is shown in Figure 2-3.

Many speech recognition systems use some variation of *dynamic programming* (DP) [13], which involves the stretching and shrinking of the templates to find the minimal distance. This is illustrated in Figure 2-4. DP is known to produce better results than linear warping [14]; however, it is far more time consuming than linear warping.

In order to obtain accurate results while reducing recognition time, a hybrid warping algorithm that

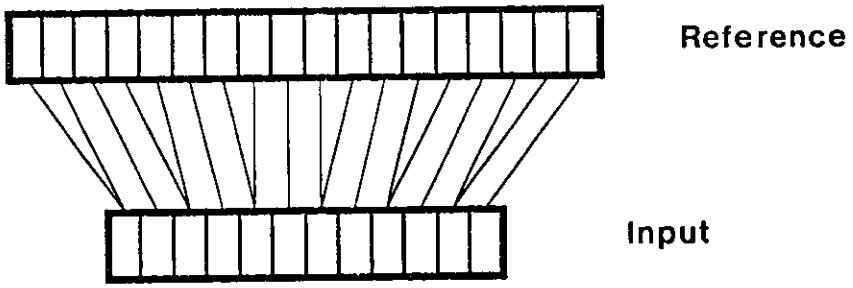

Figure 2-3: A linear warp frame alignment.

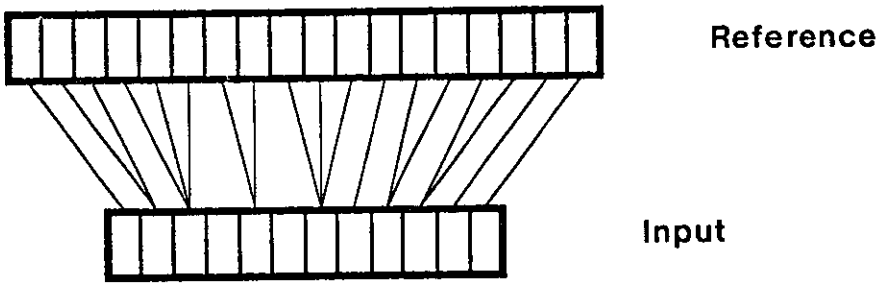

Figure 2-4: A possible dynamic warp frame alignment.

combines both techniques is formulated. This algorithm utilizes the segment boundaries produced in the segmentation phase. Using this algorithm, speech segments are *dynamically aligned* into groups; within each group, the frames are *linearly aligned.* In other words, we are looking for the way to group together the segments of the input with the segments of the reference such that the distance between the input and the reference is minimal. This is shown in Figure 2-5. We will use the term **alignment** to indicate a particular way to align the segments of the two words, such as Figure 2-5; and we will use **group** to mean a grouping of segments from the two words. An **alignment** consists of one or more **groups.**

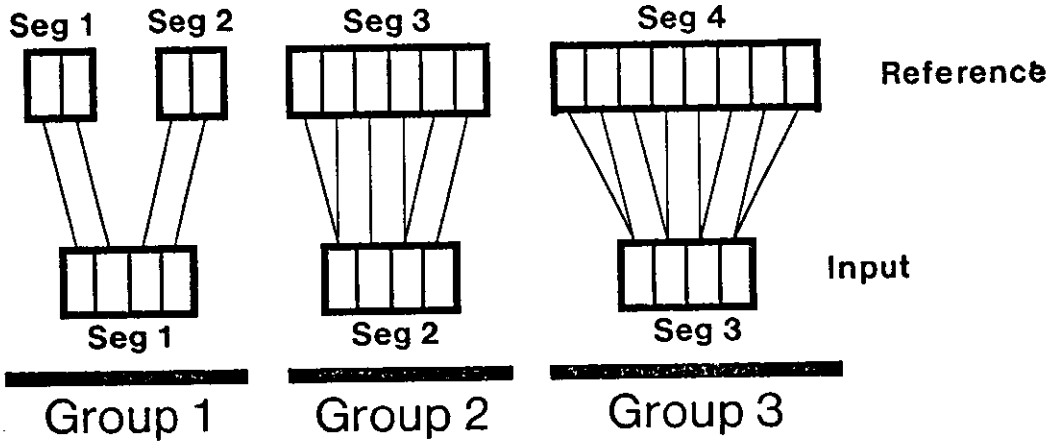

Figure 2-5: A hybrid warp frame alignment.

The above algorithm still may result in many improbable alignments. Thus, we impose four constraints upon this algorithm:

1. **Sequential Constraint** - Both utterances are processed beginning to end, with the exception of allowing skipping of segments up to 10 frames (30 milliseconds) in the beginning or the end (with some penalty).
2. **Alignment Constraint** - All groups are one (segment) to many (segments), or one to one. No many to many mappings are allowed.
3. **Durational Constraint** - Two groups of segments can be aligned only if the following relation holds for L_1 (total group length from utterance 1) and L_2 (total group length from utterance 2):
$$L_1 < (L_2 \cdot 1.75 + 10) \text{ and } L_2 < (L_1 \cdot 1.75 + 10)$$
4. **Phonetic Constraint** - It is not permissible to match any segment against segment(s) that have a large percentage (set at *50%*) of frames with a different coarse phonetic label.

Constraints 1 and 3 are similar to boundary and slope constraints in DP [15]. Constraint 2 is necessary because if we consider many to many mappings, many more alignments must be considered, and the contribution of segmentation is reduced. Constraint 4 is simply an exploitation of information available from segmentation.

Figure 2-6 shows two examples of the letter X, and their Zapdash segmentation. Figure 2-7 shows the number of possible alignments after applying constraint 1. Each rectangle shows a possible alignment; the circles represent the segments; and each partition is a group of the alignment. For example, alignment 14 aligns segments 2 and 3 of the first X with segment 1 of the second X; segment 4 of the first X with segments 2 and 3 of the second X. Segment 1 of the first X is ignored by not aligning it with any segments in the second X. The effect of constraints 2, 3, and 4 are shown in Table 2-2.

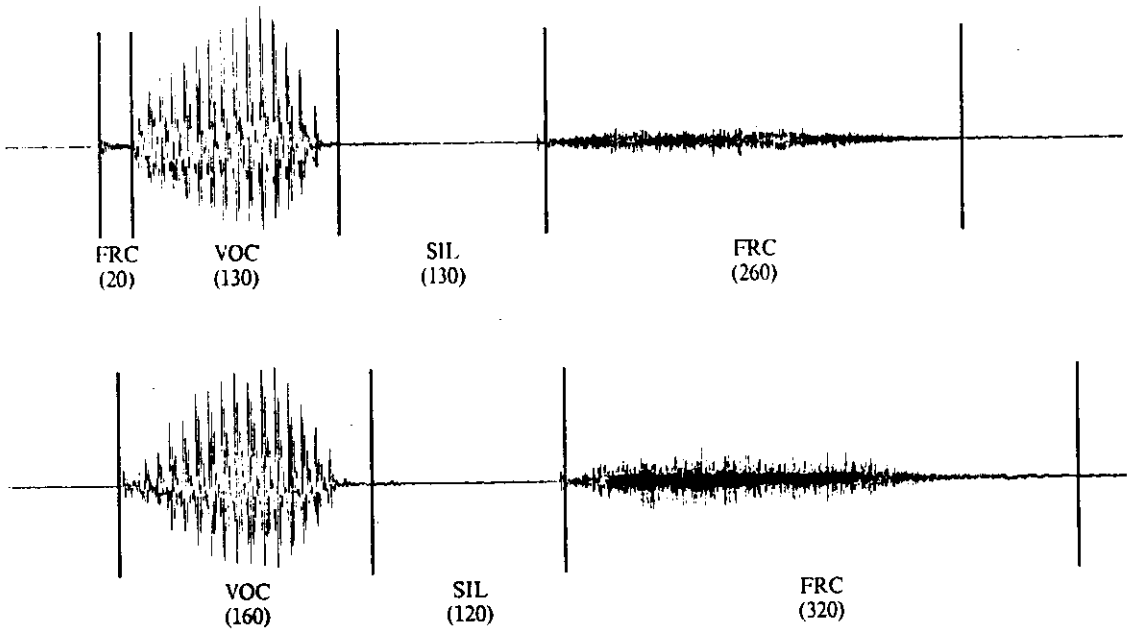

Figure 2-6: Zapdash segmentations of two examples of the letter X.

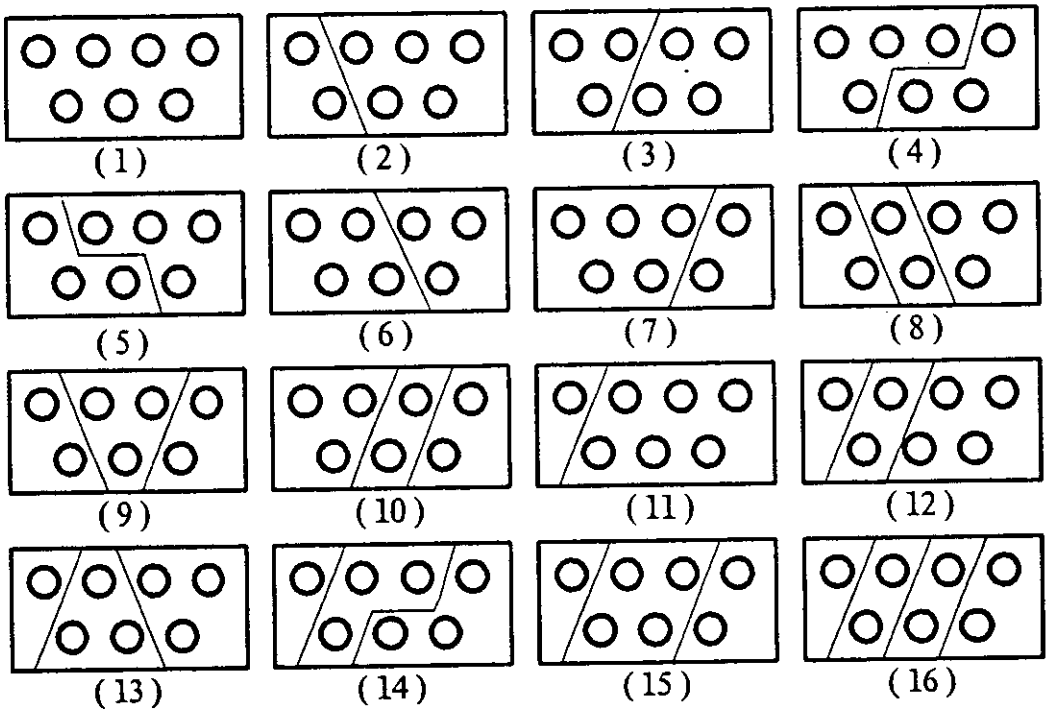

Figure 2-7: Possible alignments of the 2 X's in figure 2-6 after constraint 1.

After applying all 4 constraints, we can see that alignments 10 and 16 are indeed the only plausible alignments of the two X's. Given alignments 10 or 16, the DP should not outperform the Hybrid Warp since DP is likely to have similar frame to frame alignments.

After Applying Constraint	Alignments Not Pruned
1 *Sequential*	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
2 *Alignment*	4 5 8 9 10 13 14 16
3 *Durational*	4 10 14 16
4 *Phonetic*	10 16

Figure 2-2: The constraint propagation process in the hybrid warp.

These constraints are powerful enough to reduce the search to significantly less than that required by DP, yet are sensitive enough to produce results comparable to DP. In order to demonstrate this, the Itakura asymmetric warp [15] and linear warp were implemented within the system structure. For this experiment, the *Matching Approach Network Training* (see Section 3.2) was applied to speaker-independent and speaker-dependent alphabet data. The results in Table 2-3 demonstrate that the Hybrid Warp Procedure achieves accuracy very close to DP, yet uses only slightly more time than linear warping.

	Speaker Dependent		Speaker Independent	
Warp Procedure	**% Recog.**	**Sec./Recog.**	**% Recog.**	**Sec./Recog.**
Itakura Warp	99.23	4.80	92.20	35.49
Linear Warp	96.92	0.72	89.96	2.99
Hybrid Warp	99.23	0.78	91.92	3.57

Figure 2-3: Comparison of warping algorithms using matching approach network generation and the alphabet database.

When each frame is considered a segment, the outcome of Hybrid Warp is identical to dynamic programming; when the utterances are one segment long, it is identical to linear warp. Thus, while the hybrid warping algorithm is able to cope with considerable segmentation errors, it does rely on segmentation to acquire the power of DP. Nevertheless, when segmentation is inaccurate, one should expect degraded, but not unreasonable results. Appendix I describes the Hybrid Warp in more detail.

2.4.3 Search

The recognition phase is a search for the reference pattern which best matches the input. The search technique used here is a *branch and bound exhaustive search.* [16] Although other systems [8] [1] have performed well without searching exhaustively, the word recognition system here is sufficiently fast, and the vocabulary sufficiently small, for exhaustive search to be feasible. The speed of the system is further enhanced by branch and bound pruning. Simply stated, branch and bound means: whenever the distance of

a partial alignment has exceeded the *current minimum complete alignment distance*, it could not possibly be the best match; therefore, there is no need to complete the matching and distance computation for this alignment. After the computation of each frame-wise distance, we check if the current minimum distance has been exceeded. If so, abandon the current alignment. It is important to understand that the use of branch and bound affects only the speed, not the accuracy, of recognition.

In order to illustrate the savings of branch and bound pruning, we modified the recognition routine to search exhaustively. The savings of branch and bound is shown in Table 2-4 were obtained by *Matching Approach Network Technique* (see section 3.2). The time given is seconds per recognition. This experiment shows that branch and bound requires only 20-25% of the time used by exhaustive search.

Search Algorithm	SECONDS PER RECOGNITION OF ALPHABETS	
	Speaker Dependent	*Speaker Independent*
Exhaustive Search	3.18	16.57
Branch-and-Bound	0.78	3.57

Figure 2-4: Savings by branch and bound using matching approach network generation.

3. Network Training

Using the network representation, the reference set is encoded as a network of *nodes* linked with *paths*. A *node* is a basic unit of speech, and is equivalent to a *segment*. A *path* is a sequence of one or more nodes, and corresponds to a *word*. For example, a path for "B" could be represented as in Figure 3-1, and a network with three paths ("B", "D", and "V") and 4 nodes (/b/, /d/, /v/, and /i/) is shown in Figure 3-2.

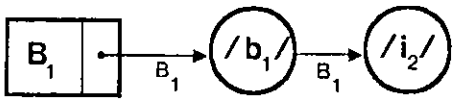

Figure 3-1: A path for "B" using the network representation.

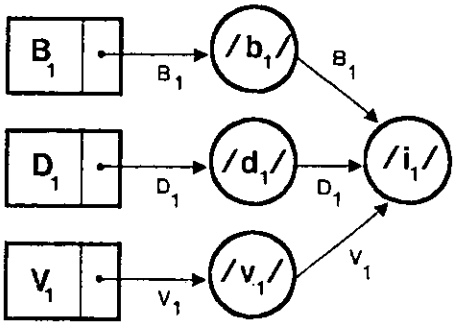

Figure 3-2: A network of 3 paths.

The network in Figure 3-2 is very effective and compact. If the three words were stored as three templates, when we match them against an input, random variations in the /i/ part will dominate the total distance because the vocalic segment is much longer than the fricative. With the above network, however, the /i/ segment of an input B, D, or V will have the *same* match score against each of the three paths. This between-word generalization leads to the ability to discriminate important (/b/ /d/ /v/ here) from unimportant (/i/ here) segments. Furthermore, the above network requires less than half the space of three full templates because two long /i/ segments are eliminated. The savings could be more significant with larger vocabulary and larger number of training data per word.

It is easy to see the advantages of the network representation; however, it is far from clear how to generate such a network from sets of training data. In this chapter, we examine two methods of network generation. Unlike traditional network systems, we use an incremental network approach. The first technique, the *learning* approach, is modeled after NEXUS [8] with some modifications. It tries to recognize each training data, and is positively reinforced for correct recognition, and learns from its mistakes. Although

this technique seems very elegant, a more compact network resulting in better recognition can be constructed using a simpler method, the *matching* approach. The *matching* approach matches words against each other to collapse similar segments. The following sections describe these techniques and the results in detail.

3.1 The Learning Approach

The *Learning Approach* attempts to recognize each token in the training data as if it were input data. Depending on the outcome of the recognition, the network is modified accordingly.

If the recognition is positive (the path in the network that best matches the input has the same identity as the input), then the system invokes the *positive exemplar* learning module. This module looks at each group in the best alignment between the input and the best matching same-word template. If every group distance is within a distance threshold, T_S, then each group of the input is averaged with its counterpart in the template. If any group distance is too large to average, a new network path is created, averaging the input with the template when the group distance is below the threshold, and using input nodes otherwise. In nearly all cases, there is no need to add another path (every group distance is below T_S).

If the recognition is negative (a word template with identity different from the input was determined to be the best match), the learning process is more complex. There are two possible causes for a negative recognition. The first cause is that the training token had not appeared before. In this case, we match the token against each of the existing paths, and when partial alignments are found to be similar, the corresponding segments (but never the whole word) are averaged together. Otherwise, the input segments are added.

In case of a real error (there exists at least one path whose identity is the same as the input token), the *learning module* is invoked to attempt to add a new path that is more similar to the input than the wrong-word match. This new path may use

- Segments from the input.
- Segments from the input *averaged* with segments from the wrong word match.
- Segments from the input *averaged* with segments from a same word template.

Next, we describe the exact algorithm that constructs this path.

The first step is to recover all groups that constitute the best alignment of the wrong template to the input, and of each template of the correct word to the input. Table 3-1 shows the groups and the scores for the initial network and input in Figure 3-3. (Note, however, that there could be other same-word (B) templates in the pool)

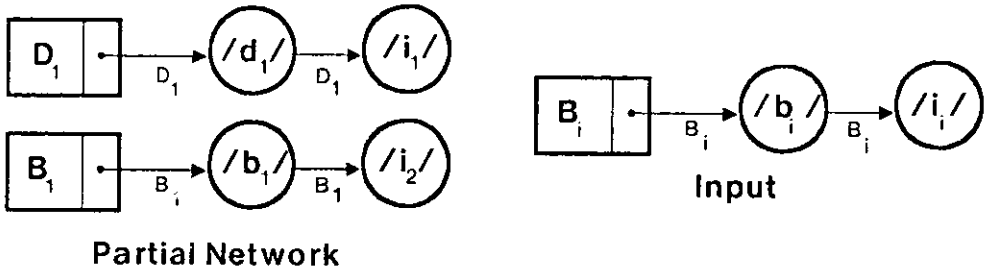

Figure 3-3: A partial network and a training input "B" before network modification.

Path	Node	/b_i/ (length = 5) Node Distance	/i_i/ (length = 100) Node Distance	Path Distance
D_1	/d_1/	90	–	2450
	/i_1/	–	20	
B_1	/b_1/	30	–	3150
	/i_2/	–	30	

Figure 3-1: Table of distances between every segment of the input and every node in the partial network in figure 3-3.

Table 3-1 shows that /b_1/ is a much better match than /d_1/ for /b_i/, and that /i_1/ is only slightly better than /i_2/ for /i_i/. However, because of the dominant length of /i_i/, template D_1 yielded a smaller distance than B_1, causing an incorrect recognition. The *network approach* tries to avoid this error in the future by adding a new path with identity B in the network. That new path must match B_i better than D_1 matches B_i.

In order to modify the network with most accuracy, we introduce a distance threshold: T_D. If any group between segments of the different word template that was incorrectly "recognized" and segments of the input has a distance greater than T_D, it is not considered as a candidate to be averaged with the input. Similarly, T_S is the distance threshold for the same word templates, and any same-word groups with distance greater than T_S are eliminated. Note that $T_S \geq T_D$ because it is never more desirable to average into a different word than into a same word. Using the above example, if $T_S = 60$ and $T_D = 30$, then (d_1, b_i) is eliminated. The pool of the remaining groups and the distance of each group (δ) are:

$$\delta(b_i, b_1) = 30 \qquad \delta(i_i, i_1) = 20 \qquad \delta(i_i, i_2) = 30$$

Next, all possibilities of combining these segments are considered, and the combination with the

minimal total distance is the new path[3]. The following shows the computation of the new path:

$$\delta(b_f, b_1)\cdot 5 + \delta(i_f, i_1)\cdot 100 = 30\cdot 5 + 20\cdot 100 = 2150$$
$$\delta(b_f, b_1)\cdot 5 + \delta(i_f, i_2)\cdot 100 = 30\cdot 5 + 30\cdot 100 = 3150$$

This indicates that $\mathbf{b_1}$ and $\mathbf{i_1}$ are the best matches for the input segments, and that they should be used in the modification of the network. Thus, the new path is the two segments: (1) $\mathbf{b_1}$ averaged with $\mathbf{b_i}$, and (2) $\mathbf{i_1}$ averaged with $\mathbf{i_i}$. Figure 3-4 shows the modified network.

In the network algorithm, when network nodes are found to match the input segments well, they are averaged together to create a more reliable template. Segments that are averaged into a network node in this fashion are weighted equally.

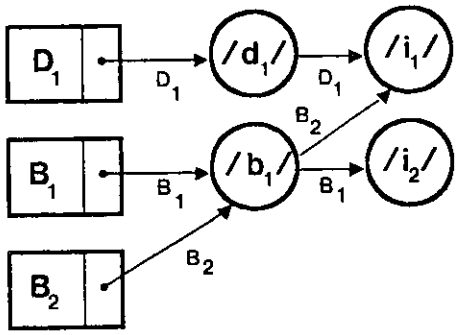

Figure 3-4: Modified network given Figure 3-3 and Table 3-1.

If $\delta(b_1, b_i)$ were equal to 80 (there is no good match for the /b_i/), then there will be no path that averages network nodes with every segment of the input. Therefore, each segment of the input is also included in the pool with the δ value of that segment set to T_S. Setting δ of each input segment to T_S results in using input segments only when it is impossible to find segments from different-word or same-word templates to average segments of the input into. In this case, the possible combinations are:

$$\delta(b_f, b_i)\cdot 5 + \delta(i_f, i_1)\cdot 100 = 60\cdot 5 + 20\cdot 100 = 2300$$
$$\delta(b_f, b_i)\cdot 5 + \delta(i_f, i_2)\cdot 100 = 60\cdot 5 + 30\cdot 100 = 3300$$

The system will correctly choose to add the input segment /b_i/, average /i_i/ with /i_1/, and add a new path with these two nodes. Figure 3-5 shows the resulting network.

The only case where this learning scheme will fail is when the new path consists *entirely* of segments corresponding to a different word (for example, if $\delta(b_1, d_1) = 20$). When this occurs, there are two possible causes:

---

[3]Actually, not all combinations are considered completely. This problem is equivalent to finding the shortest path in a directed weighted graph. Dijkstra's shortest single-source path algorithm [17] is used, and many impossible partial paths are pruned.

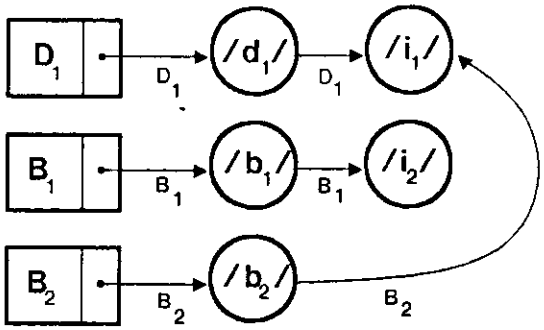

Figure 3-5: Figure 12: Modified network given Figure 10 and Table 5 when no existing nodes provide a good match to /b_1/, so that it is necessary to create new.

paths and nodes.)

1. The entire input is too similar to the incorrect different word template.
2. The entire input is too different from the templates for the same word that are already in the network.

The first case is characterized by small distance between the input and the different-word template. The second is characterized by large distance(s) between the input and the same-word template(s), and between the input and different-word template. The systems tries to decide which case is valid, and if it is the first case, the input is ignored (no adjustment of network), and if it is the second case, the worst group from the different-word template is removed from the pool and a new path is recomputed.

This concludes the description of the *learning* approach. The results and discussions of this approach will appear in later sections.

3.2 The Matching Approach

While the *learning approach* seems elegant and appeals to cognitive modeling, we believe the network generation process should be more general. Given an input, the best match for any of its segment(s) may not be part of a same-word template or the wrong-word match, beyond which the *learning* approach does not consider. For example, a template C may match an input B very poorly because of the difference in the duration of the frication; however, they may have very similar /i/ segments, which we want to average together, as shown in figure 3-6A. With the *learning* approach, no segments from C will be considered, and this results in the inferior network shown in Figure 3-6B.

In another example, the /s/ part of an input C may match the /s/ of an S template very well; however, the two /s/ sounds will never be aligned against each other in any sequential warping algorithm. The

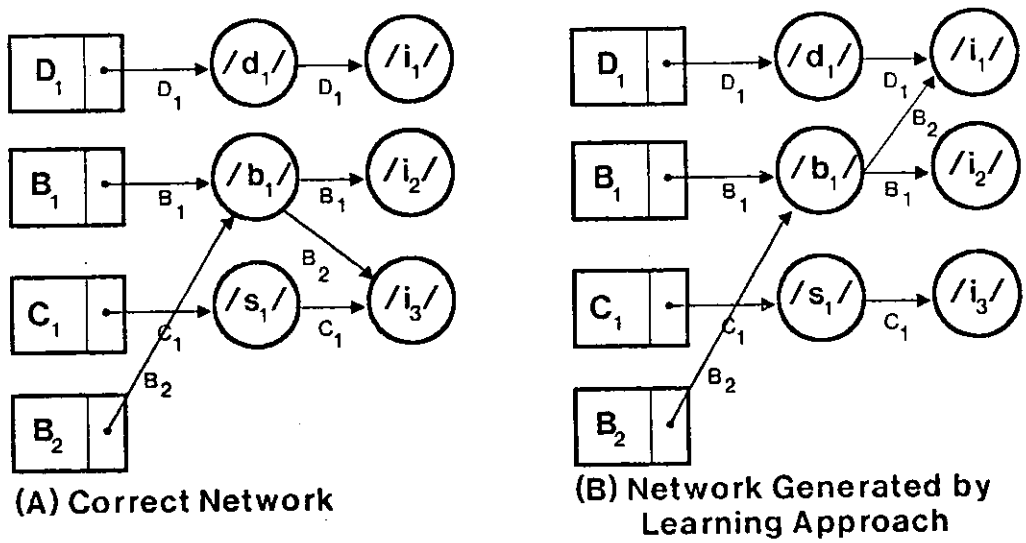

Figure 3-6: Two possible networks after modifications given the input (B_2 was added).

network should be modified as in Figure 3-7A; however, the *learning* approach will result in the network shown in Figure 3-7B.

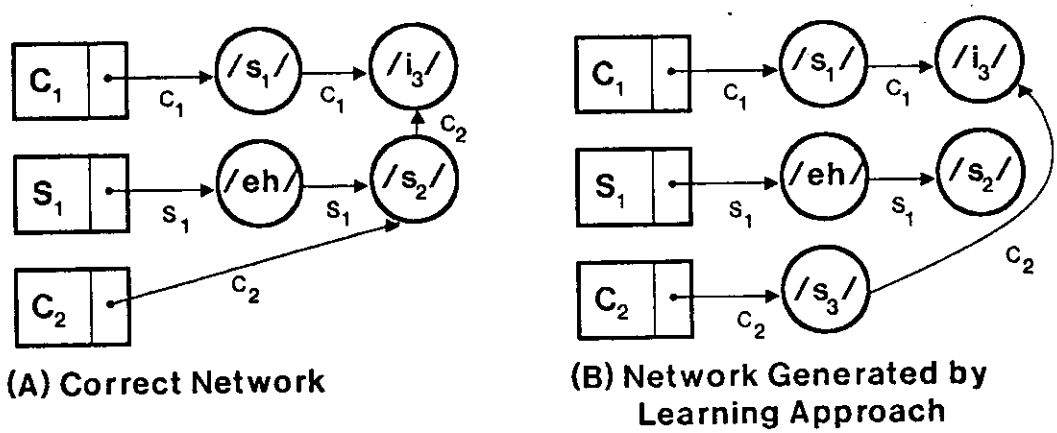

Figure 3-7: Two possible networks after modifications given the input (C_2 was added).

In each of these two cases, we would like to make generalizations, but cannot do so because of positional or durational differences between a template and the input. The *learning approach* does not allow for these generalizations. Potential candidates for the *learning approach* are summarized in Table 3-2. The *learning approach* excludes most of the templates in the network (such as Figure 3-6), and all groups in non-optimal alignment (such as Figure 3-7).

	Same-Word Templates		Different-Word Templates	
	Best Match	Others	Best Match	Others
Correct Recog.	Yes	No	No	No
Incorr. Recog.	Yes	Yes	Yes	No

Figure 3-2: Potential contributors to new path for learning approach network generation.

Thus, we introduce an alternative approach, the *matching* approach. The *matching* approach is a much simpler approach that applies the same algorithm to every token in the training set. It permits usage of groups from *any* alignment between the input and *any* reference template.

Similar to the *learning* approach, the network is built incrementally. The *matching approach* consists of two steps:

1. **Construction of a pool of groups.**
2. **Network modification.**

In the first step, every path in the current network is matched against the current training word template to identify similar groups. The Hybrid Warp constraints (Section 2.4.2) are not all applicable because somehow we must match similar sounds in different positions against each other. This can be accomplished by removing the sequential constraint in the Hybrid Warp. In other words, match any sequence of segments in the token against any sequence of segments in every template. This modified warp that computes all plausible groups between two words W_1 and W_2 is outlined as below:

```
For each segment s1 in W1
   For each segment s2 in W2
      Match s1 against s2 if other constraints are satisfied.

For each segment s1 in W1
   For each sequence of two or more segments S2 in W2
      Match s1 against S2 if other constraints are satisfied.

For each segment s2 in W2
   For each sequence of two or more segments S2 in W2
      Match s2 against S1 if other constraints are satisfied.
```

All of the matches above and their distances are saved in the pool of groups. Next, we delete from the pool: (1) each group with distance greater than T_D if the group is from a word with identity different from W, and (2) each group with distance greater than T_S if the group is from a word with the same identity as W (Note that T_D and T_S serve conveniently as branch and bound cut-off's for the group distance computation).

Then, we add each of the input segments with the group distance score equal to T_S (so that an input segment is used in the path only if no similar segment is found in the network).

In the second step, we examine all combinations of the groups in the pool, and select the best path (smallest total distance) of groups that covers every input segment once. For each group in the best path that uses network segments, we average the appropriate input segments into the network segment(s). If the group consists of an input segment only, it is used in isolation because no similar segment from the network can be used.

If the best path is equivalent to a same-word existing path, then no paths need be added. If the best path is equivalent to a different-word existing path, we either ignore the input (when it is too close to the different-word) or eliminate the worst group of the match from consideration and recompute the best path. If the best path uses segments from more than one template, they are linked together to form a new path.

Like the Hybrid Warp, this algorithm is designed to deal with inaccurate segmentation. A simpler method would be to match each segment of the input against each existing segment of the same type in the network; however, as Zapdash sometimes misses a segment, or creates an extra segment, that method would not find as many alignments that can be averaged.

Figure 3-8 shows a portion (the E-set - B, C, D, E, G, P, T, V, Z) of the actual network generated by this approach after 5 sets of speaker-dependent training data. This partial network requires less space than a single set of the 9 letters. This shows the potential for savings in space when words in the vocabulary are similar.

This completes the description of the *matching* approach. It is similar to the *learning* approach. In fact, the bulk of the two algorithms is the same, with three notable exceptions: (1) the *matching* approach makes no distinction between correct and incorrect recognition, (2) the *matching* approach allows any template to contribute to the new path, and (3) the *matching* approach permits non-sequential alignment when searching for a good match through a relaxed hybrid warp.

3.3 Results

Performance of the two network systems are displayed in table 3-3. As stated in section 2.1, the speaker-dependent results the letters of the alphabets were obtained using 5 sets of one speaker to train, and 10 other sets of the same speaker to test; and the speaker-independent results (both letters and digits) were obtained using 2 sets each of 10 speakers to train, and another 2 sets each of the 10 speakers to test.

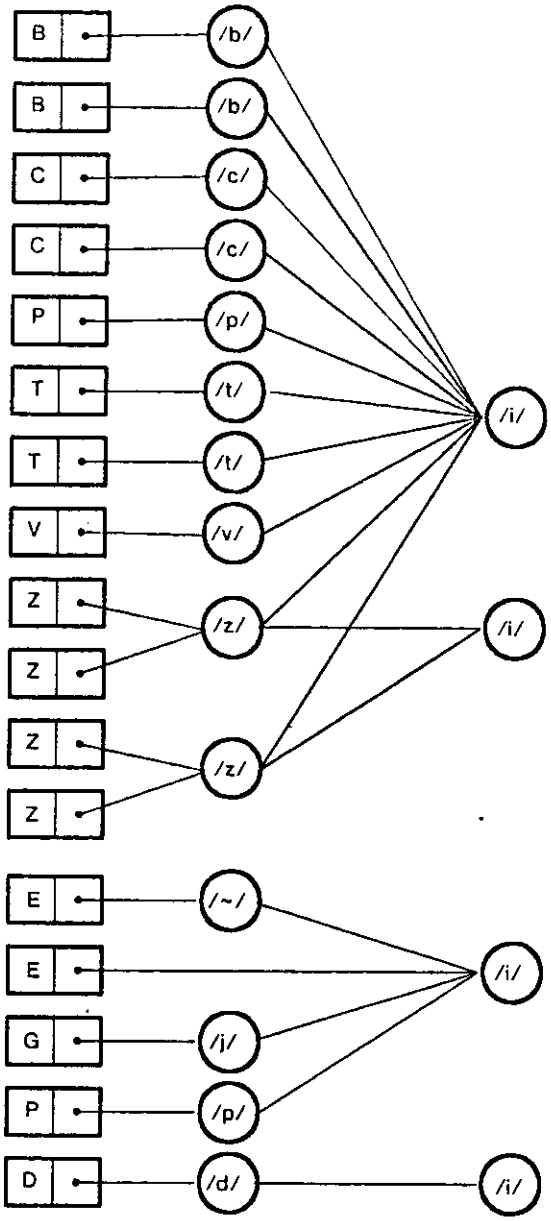

Figure 3-8: A partial network generated by matching.

"Recognition" is the percentage correctly recognized utterances out of a possible total of 260 recognitions for speaker dependent letters, 520 for speaker independent alphabets, and 200 for speaker independent digits. "Megabytes used" accounts for the data space used by the program. It is dominated by the cepstrum coefficients. Also included are the power values, network maintenance information, and the data structures for paths and nodes. "Total time for generation" is the time the network algorithm took to generate the network, and "Time per recognition" is the time used for *one* recognition.

	Speaker Dependent Alphabets		Speaker Independent Alphabets		Speaker Independent Digits	
	Learning	*Matching*	*Learning*	*Matching*	*Learning*	*Matching*
Recognition	98.08%	99.23%	89.23%	91.92%	98.00%	99.00%
MBytes used for network data	0.18	0.17	1.58	1.15	0.61	0.30
Total Time for generation (sec.)	121	93	1581	902	230	119
Time per recognition (sec.)	0.82	0.78	4.36	2.85	0.63	0.53

Figure 3-3: Recognition accuracy, space requirements, and time requirements of the two network approaches.

3.4 Discussion

Traditionally, network-based systems [1] [9] use some form of *compiled network*. Neither of the approaches described belongs in this category. They are both *incremental network* systems. Compared to the *compiled network*, the incremental approach has a number of advantages:

1. Language and vocabulary independence.
2. Easy addition of new words by learning.
3. Does not require phonetic description and alternate pronunciations of each word.
4. Does not require accurate and fine labeling.

However, it is likely to use more space and may yield inferior performance.

Nevertheless, the results in the previous section are very encouraging. In particular, the *matching* approach attained high recognition accuracy. The main reason that it produced better results is that it has a greater pool from which to select the candidates for averaging. It can be thought of as an incremental segment-level clustering process. Its superiority is predicated upon its ability to find more suitable clusters for the input segments.

Moreover, the *matching* approach is easier to implement because of its use of a uniform procedure for network incrementation. For the same reason, it is easier to understand.

The *matching* approach is superior in both time measurements. It is faster in network generation because the recognition phase in the *learning* approach could take considerable time. Also, the *matching* approach allows the use of the thresholds as branch and bound cut-off's, which eliminates the need to compare most of the dissimilar groups. Its smaller network accounts for the faster recognition time.

One disadvantage of the *matching* approach is when the vocabulary size grows larger, the *matching* approach may require a large amount of time (because of the larger pool of groups). But if better or finer segmentation were available, it may not be necessary to consider matching sequences of segments, and we could simply look for one-to-one groups. Furthermore, since reference generation occurs only ones, and the network can be easily modified later, this is not a serious problem.

One common problem to both approaches is the need of considerable space and time for speaker independent recognition. Figure 3-9 shows recognition rate vs. space usage (space usage is adjusted by tuning the distance thresholds T_D and T_S) for speaker independent alphabet recognition. The *learning* approach is clearly inferior because it requires almost over 1.5 megabytes to reach its optimal accuracy - 89.23%, which *matching* approach can attain with only 0.5 megabyte. But even the *matching* approach requires 1 megabyte (about the size of 10 data sets) for its optimal performance (91.92%). This reduces the storage requirements by only 50% from using all the training tokens.

A comparison between Figure 16 and Figure 17 shows that speaker-dependent recognition reaches asymptotic recognition performance with a much smaller amount of required storage. In fact, the asymptotic storage required only 50% more storage than what is needed for just one set of data. This is the ideal behavior that we seek. Unfortunately, a comparably efficient representation is not possible for speaker independent recognition.

One might expect recognition rate to increase monotonically with additional storage. In Figure 3-10, that is clearly not the case. Also in Figure 3-9, the slope is negative after 1 MB for the *matching approach*. The local fluctuations are caused by imperfect choice of thresholds. Our choice of T_D and T_S are not always optimal for any allowed space. An bad choice may cause a drop in accuracy while a good choice may cause an increase.

The general non-monotonic shape of the curves can be explained as follows: the variations in speech cannot be represented using one template or very few per word; thus, there is an increase in accuracy when we provide some additional storage. However, when we start adding unreliable templates, recognition rate will drop. For example, consider the case where the stabilized network has a template for **B** and another for **D**, each of which has been averaged into many times. A training input **B** that is slightly closer to **D** may be ignored or averaged if space is limited (by low thresholds). Ignoring is ideal, and averaging is unlikely to be disastrous (since averaging is weighed by number of components already averaged into the template). But if that input is added as a template; more B-D confusion will occur during recognition. Thus, recognition rate will decline with superfluous additional storage.

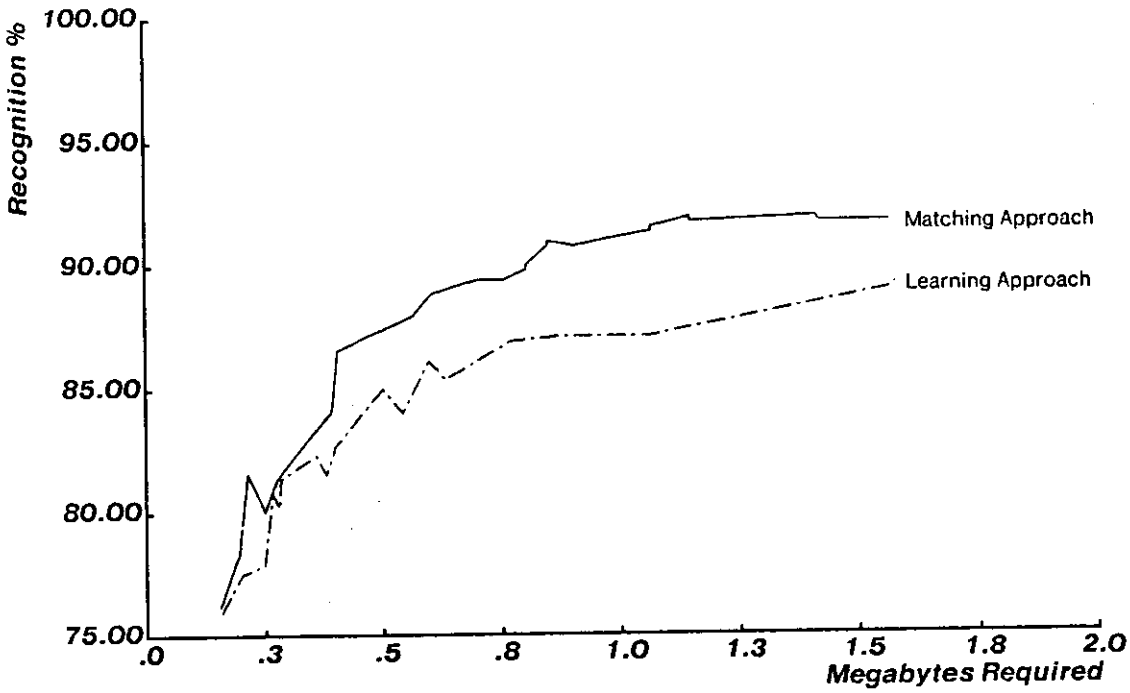

Figure 3-9: Speaker-independent recognition accuracy vs. space usage for network training.

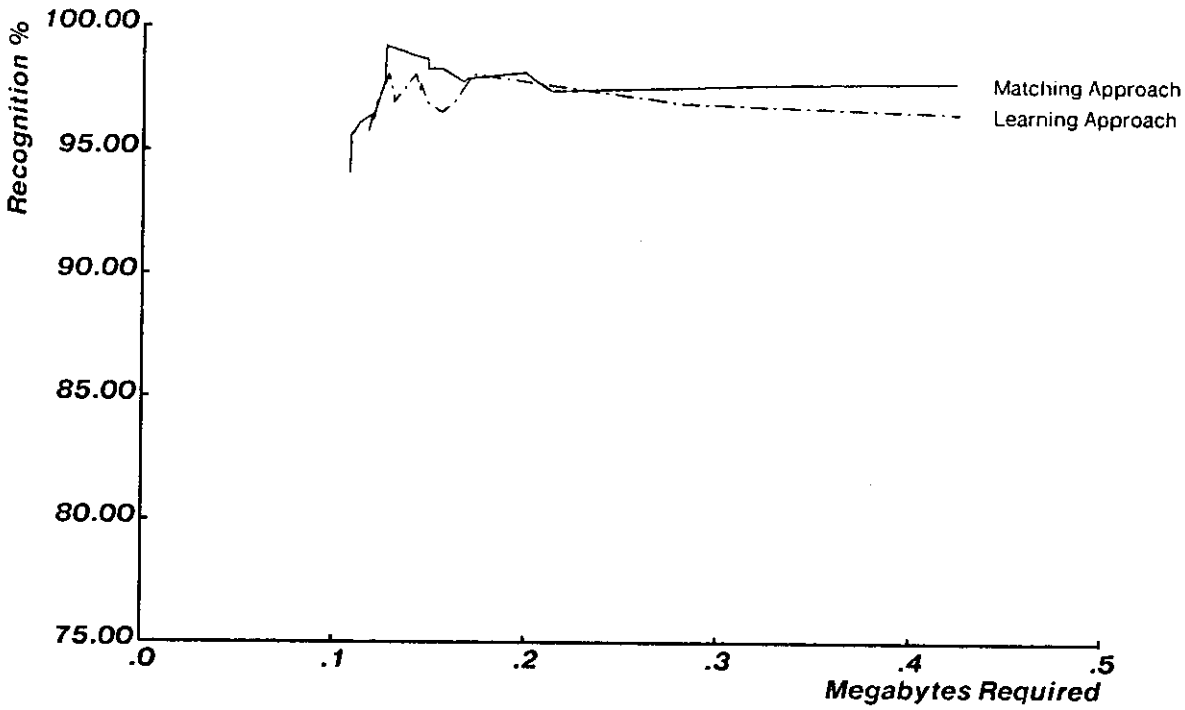

Figure 3-10: Speaker-dependent recognition accuracy vs. space usage for network training.

4. Comparison of Network Systems with Nexus

NEXUS [8] is a word recognition system using a network representation similar to the systems in this paper, and a learning strategy similar to the *learning approach*. Many underlying theories and concepts discussed in this paper were motivated by NEXUS. However, there are many differences in the representation of speech, the generation of the network, and recognition using a network-structured reference set. These differences, as well as how they effect the performance of the systems, will be discussed in the subsequent sections. In this Chapter, the two approaches will be abbreviated as LEARN and MATCH.

4.1 Speech Representation

In NEXUS, spectral coefficients from Fast Fourier Transform are used to represent the speech signal. NEXUS encodes every 12 milliseconds of speech with 26 8-bit coefficients. In contrast, the word recognition system in this study encodes every 3 milliseconds of speech with 24 9-bit LPC cepstrum coefficients. Both NEXUS and the system in this study use the Euclidean distance to measure the distance between two frames of speech. In addition, we also use power and duration information to enhance the distance measurement. Although it has been shown that cepstrum coefficients are more accurate than FFT coefficients [18], and NEXUS uses less storage, these factors cannot completely account for the difference in performance to be shown later.

Segmentation in NEXUS is based on a segmenter written by Bradshaw for NEXUS [8]. The segmentation algorithm we used is the Zapdash segmenter. The Bradshaw segmenter uses parameters quite similar to Zapdash, but uses a different strategy for segmentation. The details of that segmenter is beyond the scope of this paper. Nevertheless, it should be noted that Bradshaw's segmenter produces four broad category segment labels: **Vocalic, Fricative, Silence,** and **Unknown**; thus, another difference between it and Zapdash is that unlike Zapdash, it does not try to label segments with uncertain identity.

In a recent evaluation on a large database from multiple speakers, Waibel [19] showed that Zapdash is somewhat better, yet takes less time than the Bradshaw segmenter in classification of vocalic, silence, and fricative when compared to hand segmentation. This is the case even if all the **Unknown** segments (very few) are classified correctly by the Bradshaw segmenter. From examining many outputs of both segments, it appears that Zapdash is a more robust segmenter, and the Bradshaw segmenter worked well for the two speakers it was trained on. For these two speakers (the only two used in NEXUS), the Bradshaw segmenter was as good as, if not better than, the Zapdash segmenter. Therefore, in spite of Waibel's results, we cannot attribute the inferior performance of NEXUS to segmentation.

4.2 Network Representation

Although NEXUS has more complex data types, retaining more information about the network structure and the speech, there is no fundamental difference in the representation of the network. Both systems maintain a list of paths that represent words, and each path consists of nodes that represent segments.

4.3 Network Generation

NEXUS uses a learning paradigm quite similar to LEARN to generate its network incrementally. For each training token, NEXUS first tries to recognize it using the existing network, and on:

- **Correct Recognition** - Always average into the same-word token that matched. *This is similar to the* LEARN, *except the* LEARN *allows for adding instead of averaging segments that are not really good matches.*
- **Unknown Word** - A new path is added to the network, averaging segments into well-matching network nodes, or adding the segments themselves. *This is identical to both* LEARN *and* MATCH.
- **Incorrect Recognition** - NEXUS computes a *similarity analysis* by:
    - Find the network paths with the same identity as the input, and identify the best same-word match.
    - Match *all* network paths that have the same identity as the input word against the input word and the best same-word match.
    - Match *all* network paths that have the same identity as the incorrect best-match against the input word and the best same-word match.
    - Sum the above distances as two vectors, one of which shows a frame-by-frame total distance between the two different words, and another between different examples of the same words. Then, the two vectors are subtracted to produce a *profile*. At any frame of the profile where the value is positive, the between-word match is less similar than the within-word match.

  If either no matching same-word path is found or the profile does not correspond with segmentation, the input is added. Otherwise, parts of the input may be averaged into the best same-word match, the incorrect best match, or simply added. LEARN*'s error recovery process also locates good segments to average, but it does not need to make this correspondence.* MATCH *uses a totally different strategy.*

Furthermore, NEXUS uses *network maintenance heuristics* to eliminate "unnecessary" or "harmful" paths. An "unnecessary" path is one that has not been used to correctly recognize an input in five recognition cycles. A "harmful" path is one that was incorrectly used over 40% of the time. *The systems in this study are not equipped with these heuristics.*

4.4 Network Recognition

The greatest difference in network recognition is that NEXUS does not distinguish recognition from generation. With NEXUS, a recognition is always followed by learning and network adjustments. On the other hand, learning always requires recognition first. LEARN is similar, except for practical purposes, we have separated generation from recognition. MATCH does not require recognition in the generation stage.

Another difference is in search. NEXUS uses beamsearch, which expands potential paths in parallel. However, because of the similarity in the vocabulary, the beam size is set at a large value to minimize pruning of the correct path. Because of this, the exhaustive search used by the systems here actually uses less time for search than the beamsearch in NEXUS.

Although NEXUS segments input speech, that information is not used in the matching. Instead, it uses a standard DP matching algorithm to match the input word against every path in the network. Although DP is slightly more accurate than the Hybrid Warp, it is very time consuming. Furthermore, useful information from segmentation was not used.

4.5 Comparison of Results

Nexus was tested on speaker-dependent alphabet data of two speakers. The first half of the same data for one of these speakers (mgb) was used throughout this study as the "speaker dependent alphabet database". It is, therefore, possible to directly compare the performance of NEXUS and the systems here.

Bradshaw [8] used 30 sets of the mgb speaker-dependent alphabets to generate a network, and reported results for every 5 sets in a continuous 30-trial experiment (Recall that NEXUS does not distinguish between training and testing). Table 4-1 shows the performance of NEXUS:

Trials	Recognition %
2-5	90.38%
6-10	94.61%
11-15	93.84%
16-20	93.84%
21-25	95.38%
26-30	90.76%

Figure 4-1: Nexus recognition results.

The systems in this study used sets 1-5 to train, and 6-15 to test, and LEARN attained 98.08% recognition accuracy and MATCH reached 99.23%. In contrast, with continuous learning, NEXUS recognized 94.23% of the alphabets in sets 6-15.

In terms of storage, each system used somewhat less than the size of 2 full sets of the alphabets, although NEXUS required considerably fewer bytes to represent each set. No time measurements for NEXUS are available, but based on experience, it is much slower than the systems here because of DP matching.

4.6 Summary of Differences

Table 4-2 displays the differences and similarities between NEXUS and the two systems described in this paper.

	Nexus	Learn	Match
Representation			
Speech	FFT Coefficients	Cepstrum Coefficients	Cepstrum Coefficients
Network	Path = Word Node = Segment	Path = Word Node = Segment	Path = Word Node = Segment
Algorithms			
Distance Measurement	Euclidean	Euclidean + PD	Euclidean + PD
Segmentation	Bradshaw	Zapdash	Zapdash
Warping	DP	Hybrid Warp	Hybrid Warp
Search	Beam Search	Exhaustive Search	Exhaustive Search
Network Generation			
General Strategy	Incremental Learning	Incremental Learning	Incremental Modification
Different Situations	Correct Recognition Incorrect Recognition Network Maintenance	Correct Recognition Incorrect Recognition	Uniform Method
Options Available	Add, average	Add, average, ignore	Add, average, ignore
Number of Network Nodes Usable[4]	Few	More than NEXUS	All
Recognition %	94.23%	98.08%	99.23%

Figure 4-2: Summary of differences between NEXUS, LEARN, and MATCH.

4.7 Discussion

Although NEXUS, LEARN, and MATCH are all incremental network systems, they have many fundamental differences. The most salient one is the motivation behind each system. NEXUS was designed to substantiate the *Property Integration Theory* of speech learning. This theory of the development of human

[4] For averaging into the input. See Table 3-2.

speech perception postulates that speech is learned by integrating properties from examples. On the other hand, the systems here were designed with the goal of creating an accurate and efficient computer recognizer of human speech.

NEXUS makes no distinction between learning (generation) and recognition. Every learning trial requires recognition first, after which the network is adjusted depending upon the outcome of the recognition. This was intended to model human learning, and can (1) learn new words at any time, and (2) continuously improve perceptual accuracy. This concept was partially inherited by LEARN, but not used at all in MATCH. However, LEARN and MATCH both distinguish learning from recognition. This is because it was felt that (1) learning new words at recognition time is not an important concern for a small and novel system, and can be implemented easily, (2) results indicate that prolonged learning does not improve accuracy, and most importantly, (3) continuous learning is very time consuming; thus, it contradicts our goal of an efficient system.

LEARN inherits the NEXUS strategy of applying different algorithms depending upon the outcome of recognition. When recognition is positive, LEARN and NEXUS only differ in that LEARN can avoid using the best same-word match. This is a very minor improvement. All three systems behave similarly when presented with a new word (add the word using existing nodes whenever possible).

However, NEXUS and LEARN are quite different in their strategies of error-recovery from true recognition errors. NEXUS creates a profile by making a large number of comparisons, and then corresponding that with the segmentation to find where the correct word and the incorrectly recognized word differ. Next, it averages the segments of the input with any same-word node that is a best-matching node for that segment. Then, it uses nodes from the mismatched path if the corresponding segments were determined to be similar (Similarity is measured by a threshold value). Finally, segments from the input may be added. If either the profile or the correspondence is unsuccessful, the input is simply added.

The error recovery mechanism in LEARN has a more local view. Instead of attempting to find where the two *words* differ, it tries to find good *segments* in either the mismatch, or in any same-word path to average into, and, failing that, input segments are added. Instead of searching "best matching" node for each node only as in NEXUS, LEARN uses Dijkstra's algorithm that examines the entire space of segments from usable words. This is more likely to find a better averaging path. Furthermore, no correspondence need be created. In LEARN, the correspondence is implicit in the alignment of segments in the Hybrid Warp. Therefore, it does not suffer from the inability to find a correspondence.

However, because of the locality of its analysis, the error recovery procedure of LEARN is more likely

to average two different sounds together. If two different sounds have a lower distance than the same sounds, and that distance is below a threshold, they will be averaged together. (Note that the learning mechanism knows the identity of each word, but not that of each segment). Nevertheless, by setting T_d at a sufficiently low level, this can be minimized. Furthermore, NEXUS is prone to the same problem in the early stages of network generation when it must rely on the information of only very few paths per word.

In terms of compactness and efficiency, LEARN is superior to NEXUS. In its error recovery process, NEXUS has a tendency to:

1. Fail to find a correspondence.
2. Add the entire token.
3. The created path is pruned later by network maintenance heuristics.

The net result is a wasted training token, and deceleration of the learning process. This anomaly is avoided by LEARN and MATCH with the implicit correspondence in the Hybrid Warp.

Unlike either NEXUS or LEARN, MATCH has a generation procedure well characterized by its name. For each input, it simply finds network nodes that match its segments well, and determines the optimal way to link them (or segments of the input when nothing matches well) together using Dijkstra's algorithm. Because of the greater number of candidates considered, it is likely to build a more compact network, although it too may average different sounds together.

One feature found in NEXUS, but not in the two systems here is network maintenance heuristics. NEXUS prunes paths that have not been used for successful recognition in 5 sets of training trials, and paths that have over 40% misidentification rate. For a speaker dependent system, the pruned paths are likely to be unusual or ambiguous tokens; however, we believe that this cannot be generalized for speaker independent recognition. In speaker independent recognition, many paths may be unsuccessful or unused for some speakers, yet very successful and used often for other speakers. Pruning of these paths will cause degraded performance. But since NEXUS was intended to be a speaker dependent system, these heuristics were helpful to reduce the amount of computation.

NEXUS uses full dynamic programming matching of templates with beam search. However, because of the inherent ambiguity of the alphabets, the size of the beam must be set at a large value to minimize the pruning of the correct path. Thus, NEXUS takes substantially more time than LEARN or MATCH even with the network maintenance heuristics. This is another impediment in extending NEXUS to speaker-independent recognition.

The Hybrid Warping Procedure introduced in this paper is a good compromise between linear and

non-linear alignment. It is able to achieve accuracy close to DP, yet reducing the amount of computation considerably. It is an important factor in making the network systems here almost real time. With the Hybrid Warp, it is not necessary to make correspondences between the alignment and the segmentation, which avoids the loss of information when correspondence cannot be found. MATCH uses four times as much storage as NEXUS for speech data, and uses exhaustive search rather than beam search. Yet it is much faster and considerably more accurate than NEXUS. This is largely due to the use of the Hybrid Warp instead of dynamic programming.

In conclusion, although the parametric representation and the distance measurement were helpful to LEARN and MATCH, it is the Hybrid Warp, the reliable segmentation, and the new algorithms that enabled LEARN and MATCH to produce the superior results. In particular, the simple algorithm in network generation used by MATCH produced compact, efficient, yet accurate networks.

5. Comparison with Other Techniques

In the last two chapters we considered the theoretical soundness of network training and compared the performance of the *learning* and the *matching* approaches. In this chapter, we assess the practical soundness of network training by comparing its results with several known techniques. Each subsequent section discusses a technique, leading to the final section which compares the performance of these systems in terms of recognition, space, and time. In order to make valid comparisons, all of the techniques were implemented using the same word recognition system described in Chapter 2. In many cases, the performance of these systems depend upon selecting a number of thresholds. The results displayed reflect those obtained with the optimal thresholds. These methods are described and discussed in more detail in Appendix III.

5.1 Casual Training

Casual Training is the simplest and most natural reference generation technique. It uses one or more sets of training data as the reference set without any processing. Both *Single-Template Casual Training* (1 set of training data as template) and *Multiple-template Casual Training* (5 sets of training data were used for speaker-dependent recognition, and 20 sets were used for speaker-independent recognition) were implemented.

5.2 Template Selection

The *Template Selection* technique selects the "best" template for each word from the available training sets and uses it as the reference [3]. The basic algorithm is as follows:

1. Match every word in the training sets against every other word.
2. Select 2 best candidates for each word based on minimization of distance against same words and maximization of distance against different words.
3. Examine all combinations among them to find the best template for each word.

5.3 Averaging Training

Averaging Training takes different tokens of the same word and averages them together to create reference templates. The averaging technique used here averages *all* of the tokens together (assuming that the Hybrid Warp is able to align them). These tokens are averaged into the token whose duration is closest to the average duration.

5.4 Simplified Clustering Training

Simplified Clustering Training [2] begins with two tokens of each word. If the distance between the tokens is below a distance threshold, they are averaged to create the reference template for that word, and the generation process for this word terminates. Otherwise, another token of that word is introduced, and it is matched against all examined tokens for that word. This process continues for each word until either a pair with distance below the threshold is found, or if we run out of training data, in which case the first template is added. The tokens are processed in order of durational proximity to the average duration of all tokens with the same identity.

5.5 K-Means Clustering

For speaker-independent recognition, we have sufficient templates for statistical clustering. The algorithm selected here is *K-Means Iteration* [6]. An outline of the algorithm (to be iterated for each word in the vocabulary) is as follows:

1. **Initialization** - Select N (optimal = 3 for speaker-dependent and 9 for speaker-independent) random templates as initial cluster centers.
2. **Classification** - For each replication of the word, find the cluster center that is closest to it, and categorize that replication in that cluster.
3. **Recomputation** - Recompute the center for each of the N clusters. The cluster center is the template in the cluster that has the smallest maximal distance to any other template in the cluster.
4. **Convergence Test** - If each center remains unchanged, we're finished with this word. Otherwise go back to step 2.

5.6 Comparison

5.6.1 Results

Table 5-1 shows the recognition results of all 8 systems. All experiments were run with the database described in Section 2.1, and the recognition system description in Chapter 2. Table 5-2 shows the space requirements for speech data in megabytes required by each of the systems. This includes the coefficient, power, and data structures. Table 5-3 shows the total time for reference generation. This is the CPU seconds used to generate the entire reference set. More importantly, it also shows the time required for each recognition given the generated reference set.

Training Method	Speaker Dependent Alphabets	Speaker Independent Alphabets	Speaker Independent Digits
Casual *single template*	91.53%	52.30%	74.50%
Casual *multiple template*	96.92%	89.23%	97.00%
Selection	94.23%	60.21%	82.00%
Averaging	96.92%	65.77%	90.50%
Simplified Clustering	95.38%	57.88%	82.00%
K-Means Clustering	98.08%	90.19%	98.00%
Network *learning*	98.08%	89.23%	98.00%
Network *matching*	99.23%	91.92%	99.00%

Figure 5-1: Recognition accuracy for 8 systems.

Training Method	Speaker Dependent Alphabets	Speaker Independent Alphabets	Speaker Independent Digits
Casual *single template*	0.12	0.11	0.04
Casual *multiple template*	0.59	1.93	0.69
Selection	0.12	0.11	0.04
Averaging	0.12	0.11	0.04
Simplified Clustering	0.12	0.11	0.04
K-Means Clustering	0.35	0.95	0.34
Network *learning*	0.18	1.58	0.61
Network *matching*	0.17	1.15	0.30

Figure 5-2: Network size (in megabytes) for 8 systems.

	Speaker Dependent Alphabet		Speaker Independent Alphabet		Speaker Independent Digit	
Training Method	Total Gen. Time	Time per Recog.	Total Gen. Time	Time per Recog.	Total Gen. Time	Time per Recog.
Casual *single template*	2	0.34	2	0.40	1	0.17
Casual *multiple template*	11	1.45	55	6.17	14	1.70
Selection	654	0.40	3704	0.53	370	0.16
Averaging	38	0.32	139	0.55	47	0.15
Simplified Clustering	11	0.32	13	0.51	5	0.16
K-Means Clustering	49	0.72	424	2.15	111	0.42
Network *learning*	121	0.82	1581	4.36	230	0.63
Network *matching*	93	0.78	902	2.85	119	0.53

Figure 5-3: Time requirements (in seconds) for 8 systems.

5.6.2 Discussion

The techniques *Single-Template Casual Training*, *Selection*, *Averaging*, and *Simplified Clustering* are single-template techniques. As such, they perform satisfactorily on speaker-dependent data because a single template can represent the characteristics of one speaker reasonably well. However, in the speaker independent case, their recognition accuracies are unacceptably low because between-speaker differences are too diverse to be encapsulated in one template.

The four multiple-template techniques - *Multiple-Template Casual Training*, *K-Means Clustering*, *Learning Network*, and *Matching Network* performed reasonably for both speaker-dependent and speaker-independent data. Among the four, the *matching* approach of network generation stands out as the best.

In the introduction, we conjectured that *Multiple-Template Casual Training* is very wasteful, that *Clustering* reduces much of this wastage, and that *Network* representation increases the ability to discriminate. The results in the previous section confirm this conjecture. *Clustering* yielded better results than using all templates, and also used less storage and time. *Network* (matching) yielded results even better than *Clustering*, and could potentially use less space (by sacrificing some accuracy).

In summary, the results in this chapter showed that:

1. Single-templates cannot capture between-speaker differences.
2. For speaker-dependent recognition, single template systems (in particular, *Averaging*) are adequate, and multiple template systems are not needed.
3. For speaker-independent recognition, clustering adds the ability to generalize. It has a more compact representation than using all of the templates, yet attains higher accuracy.
4. For speaker-independent recognition, the network representation provides the ability to discriminate through the generalization of similarities between tokens with different identity. Thus it achieves the best results.

6. Future Work

Although the *matching* approach has produced satisfactory results, there are still a number of problems:

Recognition One obvious method to improve recognition accuracy of the system is to incorporate phonetic knowledge into the system. We could label the segments, and average only those with the same label. This need not be done manually, but can be done automatically with a dictionary of plausible segmentations of each word. This, however, would sacrifice the property of vocabulary independence for higher accuracy. Another possibility is to have finer segment labels so that the Hybrid Warp can yield accuracy even closer to dynamic programming. It is possible to produce segmentation with 5 or 6 labels with reasonable accuracy. We will try to use a more complete set of the Zapdash [7] outputs. Hopefully, this will cause the network to be more compact, as well as yield better results.

Space As Figure 3-9 illustrates, the network requires much space in order to achieve high accuracy for speaker-independent recognition. In view of the small vocabulary and number of speakers, this could become a serious limitation when applied to a large vocabulary. The cause of the inability to build a compact network may be (1) The group of speakers is a very diverse one, (2) the segmentation algorithm yielded segments that were too long (too many one-segment words), or (3) The distance measurement or warping method is imperfect. These must be investigated. Furthermore, vector quantization is a new method that reduces the space usage dramatically. Its use with incremental network will be the subject of another study. Finally, it is also quite possible that network techniques, as well as other template-matching techniques are limited to speaker dependent recognition. If this unfortunate possibility were true, speaker adaptation may be combined with the network approach to deal with speaker independent recognition.

Time To reduce the time requirements for recognition, we could adopt beamsearch [1] or branch and bound with pruning [16], both of which look at several paths concurrently, and avoids repeated comparisons between nodes. It is not known, however, whether this will reduce recognition accuracy considerably.

7. Conclusion

In this paper, the network representation of speech is presented. It was shown that the network structure has the desirable properties of

1. Savings of time and space.
2. Encapsulation of variations in speech.
3. Ability to discriminate and generalize.

The major impediment to the network representation is its dependence on good segmentation. This dependence is minimized by:

1. Modified Zapdash segmentation with coarse phonetic labels.
2. A Hybrid Warp that tolerates imperfect segmentation.

The incremental network approach is examined as an alternative to the traditional compiled network method. Generating the network incrementally has a number of advantages:

1. Language and vocabulary independence.
2. Can add and learn new words easily.
3. Does not require phonetic description and alternate pronunciations of each word.
4. Does not require accurate and fine labeling.

Two methods of network generation are introduced. Using an alphabet database, the *matching approach* network generation attained 99% for speaker dependent alphabet recognition, 92% for speaker independent alphabet recognition, and 99% for speaker independent digit recognition. These results are superior to all other methods implemented using the same database and recognition system, confirming the abovementioned properties of the network representation and the incremental network approach.

I. The Hybrid Warp

The Hybrid Warp is a novel warping algorithm introduced in this paper. It is outlined and justified in Section 2.4.2. In this Appendix, we will describe the Hybrid Warp in more detail.

The Hybrid Warp aligns the segments dynamically, and within each group of the alignment, the frames are aligned linearly. The linear alignment of the frames is trivial; it is done simply by taking the ratio of the lengths of the two segment sequences, and corresponding each frame of the *reference* to a frame of the *input*. Thus, each frame of the reference corresponds to at least one frame in the test input while certain frames of the input may be skipped or repeatedly used.

To align the segments dynamically, we formulated a specialized DP matching procedure. Recall the example in Section 2.4.2 that matches a reference of 4 segments against an input of 3 segments. Using this example, we can generate a standard DP matrix as shown in Figure I-1. In this matrix, $M_{i,j}$ means : the best way to match the 1^{st} through i^{th} segments of the reference against the 1^{st} through j^{th} segments of the input. Thus, $M_{1,2}$ means the best way to match the first segment of the reference against the first two segments of the input, and $M_{4,3}$ is the best way to match the entire reference against the entire input.

$M_{1,1}$	$M_{2,1}$	$M_{3,1}$	X
$M_{1,2}$	$M_{2,2}$	$M_{3,2}$	X
X	X	X	$M_{4,3}$

Figure I-1: DP matrix of matching a 4-segment reference against a 3-segment input. The crossed out cells are unusable.

Each cell of the matrix can be computed as follows:

$$M_{i,j}=\begin{cases} i=1 \text{ or } j=1 & \text{Cost }(1, i, 1, j) \\ \text{Otherwise} & \text{MIN}\left(\begin{array}{ll} \text{MIN }(M_{i-1,k}+\text{Cost }(i, i, k+1, j)) & j>k\geq 1 \\ \text{MIN }(M_{k,j-1}+\text{Cost }(k+1, i, j, j)) & i>k>1 \end{array} \right) \end{cases}$$

where Cost(r_1, r_2, i_1, i_2) is the distance obtained by aligning segments r_1 through r_2 of the reference against segments i_1 through i_2 of the input.

The first column and the first row of the matrix can be calculated immediately because they satisfy the alignment constraint (matching one against many or one against one). These values of these cells could be computed as follows:

1. Reject (set to infinity) if other constraints are not satisfied. This could be due to:
   - One group is much longer than the other (*Durational Constraint*).
   - The aligned segments are too phonetically different (*Phonetic Constraint*).
   - The remaining length (after this segment) of one word is much longer than that of the other word (indirect application of *Durational Constraint*).
2. Linearly match the frames involved otherwise.

The remaining cells depend upon an L-shaped group of previous cells as shown in Figure I-2. This is because to compute the $M_{i,j}$ cell, we must use some previously computed cell, and add a legal alignment to that (for example, we could use $M_{i-1,j-1}$ and add Cost(i, i, j, j)). The choice for this previously computed cell is limited to $M_{i-1,k}$, where $k<j$, and $M_{k,j-1}$, where $k<i$. Any other cell would force a many-to-many alignment, which violates the alignment constraint.

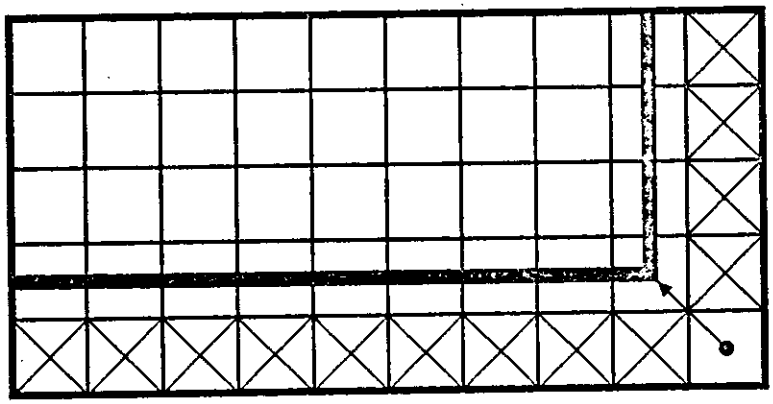

Figure I-2: Dependence of a DP cell on the other cells.

Because of this dependence relationship, except for the solution cell (the lower-right cell in the matrix), the last column and last row need not be computed since nothing depends on them. For example, $M_{1,3}$ in Figure I-1 is impossible as it leaves the segments 2 to 4 of the reference unaligned. The useless cells are crossed out in Figure I-1.

To compute the entire matrix, we first compute the cells in the first row and column. The remaining cells can be computed horizontally or vertically. Whenever an alignment is not possible due to other constraints, the value of the cell is set to infinity. Moreover, the best path distance can be used to terminate partial computations (branch and bound). Thus, in reality, many cells were never computed since they

correspond to implausible alignments. As in standard DP, the path taken is also remembered, and when the final $M_{i,j}$ is reached, the best alignment can be recovered with a backtrace.

The sequential and alignment constraints were implicitly built into this DP matching scheme. The durational and phonetic constraints were taken into consideration in the cell computation in that any cell that forces illegal durational or phonetic alignments is set to infinity.

II. System Tuning Experiments

A number of decisions were made about the word recognition system, such as the number and the size of the coefficients, the distance metric, warp parameters, and the search method. These decisions were derived by tuning thresholds and parameters and selecting the ones that optimize the recognition results. This Appendix describes these tuning experiments in detail.

It should be noted that these experiments used different data and different generation methods. But in no case was the digit database used. The digit recognition experiment was run using the parameters derived from speaker-independent alphabet recognition.

II.1 Number of Cepstrum Coefficients

24 9-bit coefficients were used initially. Later, Shikano [12] showed that 16 coefficients and 24 coefficients produced equivalent results. Thus, an experiment was conducted for five systems using 8 to 24 coefficients. In this experiment, five reference generation methods were applied to speaker-independent recognition of alphabets. The results are shown in table II-1.

Number of Coefficients	RECOGNITION %				
	Casual	Average	K-Means	Learning	Matching
8	51.15%	63.65%	87.31%	88.27%	89.81%
10	50.76%	64.04%	87.50%	88.85%	89.81%
12	51.15%	64.42%	88.08%	88.85%	89.62%
14	51.53%	64.81%	88.08%	88.65%	90.38%
16	52.30%	65.00%	88.65%	88.27%	90.58%
18	52.15%	65.38%	88.65%	88.46%	90.77%
20	52.30%	65.58%	88.27%	89.23%	91.15%
22	52.30%	65.77%	88.27%	89.23%	91.54%
24	52.30%	65.77%	90.19%	89.23%	91.92%

Figure II-1: Recognition effected by the number of cepstrum coefficients used.

These results show that the ranks of these generation techniques are preserved when the number of coefficients are varied. Moreover, they indicate that recognition accuracy varies directly with the number of cepstrum coefficients, and that 24 coefficients were necessary to retain the level of accuracy. This apparent disagreement with the results reported by Shikano [12] might be due to the fact that the original 32-bit coefficients were compressed into 9 bits in this study, while they were retained as 32-bit floating point numbers in [12]. Thus, the loss of information from the compression might be recovered by increasing the number of coefficients.

II.2 Size of the Cepstrum Coefficients

Initially, a VAX 11/750 was used for this research. On the 11/750, floating point arithmetic is considerably slower than fixed point; therefore, the 32-bit floating point cepstrum coefficients were converted to integers. There is one problem with this compression, namely, there is no theoretical upper or lower bound on the value of the cepstrum coefficients, yet any effective compression process requires tight bounds. However, in practice, almost all cepstrum coefficients were found to lie within a certain range (-2.54 to +2.54). An early experiment was conducted to compare 32-bit and 9-bit representation using casual training. This limited experiment was run using 15 sets of the speaker-dependent alphabet database. The task was recognition of B, D, V (the most confusable triple in the alphabets) using *Single-Template Casual Training*. This resulted in a 67.38% recognition accuracy for the 9-bit representation, and 67.59% for the 32-bit representation. This result encouraged the compression of the coefficients.

Another more detailed experiment compared the performance of representations of different sizes from 4 to 16 bits using five types of training for speaker-independent recognition of the alphabet. The results are shown in Table II-2. From this table, it can be seen that a 9-bit representation is a reasonable compromise. Again, the relative ranks of the generation methods are not disturbed by changing the number of bits per coefficient.

Number of Bits	RECOGNITION %				
	Casual	Average	K-Means	Learning	Matching
4	51.92%	64.96%	87.88%	88.65%	91.08%
5	52.30%	65.19%	87.88%	90.35%	91.58%
6	52.30%	65.77%	88.65%	90.35%	91.54%
7	52.69%	66.15%	88.27%	90.00%	91.73%
8	52.69%	65.77%	88.27%	89.23%	92.31%
9	52.30%	65.77%	90.19%	89.23%	91.92%
10	52.30%	65.77%	90.19%	89.23%	91.92%
12	52.30%	65.77%	90.19%	89.23%	92.31%
14	52.30%	65.77%	90.19%	89.23%	91.92%
16	52.30%	65.77%	89.23%	89.23%	91.92%

Figure II-2: Effect of number of bits per coefficient on recognition.

II.3 Distance Measurements

II.3.1 WLR vs. CEP

Initially, the WLR distance metric [12] was used. The WLR distance is defined as

$$\sum_{k=1}^{N}(C_{ik}-c_{jk})(P_{ik}-p_{jk})$$

where C_{ik} and c_{jk} represent the k^{th} cepstrum autocorrelation coefficient of the i^{th} frame of the reference and the j^{th} frame of the input, and P_{ik} and p_{jk} represent the k^{th} LPC autocorrelation coefficient of the i^{th} frame of the reference and the j^{th} frame of the input. $N (= 24)$ is the order of LPC analysis.

Sugiyama and Shikano showed that WLR is superior to CEP [11]; however, Nocerino [10] presented evidence to the contrary. Shikano [12] attributes the cause of this difference to vocabulary, claiming that WLR is more sensitive to voiced speech (more useful for Japanese recognition) while CEP is more sensitive to the unvoiced speech (more useful for English recognition).

An early experiment was conducted to compare these two distance measures. Both WLR and CEP were augmented with power and duration information, and were applied to alphabet recognition. Both single speaker and multiple speaker databases were used, but that speaker-independent recognitions were based on only 4 speakers (2 male and 2 female). Table II-3 shows the results of both *Single-Template Casual Training* and *Averaging Training*:

	Speaker-Dependent		Speaker-Independent (4 speakers)	
Distance Measurement	**Casual**	**Averaging**	**Casual**	**Averaging**
WLR + Power + Duration	91.19%	95.54%	78.54%	86.31%
CEP + Power + Duration	91.53%	96.92%	81.62%	89.43%

Figure II-3: Comparison of performance of WLR and CEP.

Recognition of the English alphabets requires far more sensitivity in unvoiced regions than voiced (consider the E-set and the Eh-set, by far the most confusing subsets of the alphabets). Thus, these results uphold Shikano's claim of the sensitivity of the cepstrum distance to unvoiced speech.

Another merit of the cepstrum distance is that it required only the cepstrum coefficients while the WLR distance required both the cepstrum and the LPC autocorrelation coefficients, which doubles the storage needed. Based on this superior performance with 50% storage, the cepstrum distance was adopted.

II.3.2 Power and Duration Weight

Table II-4 shows the results of modification of W_p and W_d, the weights multiplied by the power and duration in the computation of distance. (See Section 2.4.1) In this study, W_p was tuned first without duration information. Then, W_d was tuned with the optimal W_p.

Percent Contribution	Speaker-Dependent Alphabet Database		Speaker-Independent Alphabet Database	
	Power	Duration	Power	Duration
0%	90.00%	90.76%	47.69%	50.76%
3%	90.00%	91.53%	48.07%	52.30%
5%	90.76%	90.76%	48.85%	52.30%
10%	90.76%	90.00%	50.76%	51.92%
15%	89.23%	90.00%	50.00%	50.76%
20%	86.15%	87.69%	48.07%	47.69%

Figure II-4: % contribution of power and duration and effect on recognition accuracy.

II.4 Hybrid Warp Parameters

The Hybrid Warp requires several parameters to eliminate unlikely matches, and many choices were considered. Table II-5 illustrates the parameters and the actual choices tried:

Parameter	Choices considered
Maximum number of frames that can be skipped in the beginning and end of the utterance	5, 10, 15, 20
Penalty assigned for each frame skipped	5000, 10000, 15000, 20000
Percentage of equal-labels required in a group	10, 20, 30, 40, 50, 60, 70%
Length Differences	1.5X + {0, 5, 10, 15}, 1.75X + {0, 5, 10, 15}, 2X + {0, 5, 10, 15}, 2.25X + {0, 5, 10, 15} 2.5X + {0, 5, 10, 15}

Figure II-5: Hybrid warp parameter choices considered in deriving the final procedure.

Almost all combinations were tried using casual training to determine the final parameters. These experiments showed that the parameters did not effect recognition accuracy significantly (± 3%), but they eliminated much of the unnecessary computation.

The number of frames skipped is handled by adding null segments with the assigned length. Since the

Hybrid Warp allows for one group to be as much as 1.75X + 10 frames longer than the other group, a null segment of 10 frames allows for skipping segments up to 17.5 frames (In that case, we have a group of 10 frames null node matched against 10 frames-of null node and the skipped segment of 17.5 frames. The distance of this group is the penalty times 17.5 frames length).

Another experiment compared three ways of summing the group distances between two words:

1. Weigh each group according to its length. (Simple addition, most common)
2. Weigh each group equally. (Giving a short segment as much weight as a long one)
3. Weigh each group according to the logarithm of its length.

Results showed that the first way is the most effective.

II.5 Search

The only search implemented in this study is exhaustive full search. Although beam search or branch and bound with pruning can reduce the computation, they were not used for several reasons:

1. They may reduce the accuracy; we would like to push the accuracy to the limit.
2. Beamsearch is very difficult to implement given the Hybrid Warp with DP as described in the previous section.
3. For a small vocabulary system, the speed we achieved is satisfactory.

III. Detailed Description and Results of the Systems in Chapter 5

In Chapter 5, a number of reference generation techniques were discussed briefly. This Appendix provides additional information about these techniques, including,

- More detailed description.
- Tuning experiments. (Note that only the alphabets were tuned for speaker dependent and independent recognition; the parameters derived in the speaker independent experiments were used in digit recognition)
- Additional discussions.

III.1 Casual Training

Little more need be said about *Casual Training* because of its simplicity. *Single-Template Casual Training* installs one set of the training data as the reference set; *Multiple-Template Casual Training* installs multiple sets of templates (5 for speaker-dependent and 20 for speaker-independent) as the reference set. No processing other than this is required for network generation.

The results obtained with *Casual Training* are relatively high [20] [21]. Several factors contribute to this:

1. The use of more precise coefficients (24 9-bit coefficients for every 3 milliseconds of speech).
2. The use of cepstrum coefficient and modified cepstrum distance.
3. The use of *Casual Training* to tune the initial word recognition system (size and number of coefficients, warp parameters, and weights for the power and duration in distance calculation).

III.2 Selection

Selection is a generation technique that does not create synthetic templates. Instead, natural templates are used. There are two types of selection:

1. Consider tokens for one word at a time. Then choose the most appropriate one for that word. This is a clustering technique.
2. Consider the entire set of training tokens. Choose an appropriate token for each word.

The generation technique used in this study is that of the second type. The task is more complex than simply clustering. For each token, we must consider:

1. How alike is it to tokens of the same identity?
2. How different is it from tokens of different identities?

The selection algorithm in this study uses these two criteria to choose the final set of reference tokens. Each token is assigned a score, and a higher score indicates that it is a better representation for that word. The score for each token is calculated as follows:

- If a different-word token matches a same-word token better than this token does, subtract 1.
- If this token matches a different-word token better than another different-token does, subtract 1.

This score is, of course, only heuristic since some of the matches are immaterial. But it does provide a reasonable guideline as to which tokens cause more confusion.

To improve the final selection, the top one or two candidates for each word are saved, and all possibilities of choosing the reference set from them are considered. For each set, we compute how many of the training tokens would have been recognized correctly. Finally, the set producing the highest recognition accuracy is used.

The heuristic score computation part is not very time consuming; however, the second part which computes all possible reference sets faces combinatorial explosion. This problem, together with the inferior results, seriously limits the usefulness of the selection method.

A similar method leading to similar results is given by [3].

III.3 Averaging

In order to average a group of tokens together, we must resolve one problem - which token do we average these tokens into? If we select the token with the longest length, it is possible that an input may be too short to be matched against it. Conversely, if we select the token with the shortest length, it may not match against a long input. Furthermore, if we select a token with an "abnormal" segmentation, it may not match against an input of another "abnormal" segmentation. (For example, if the word has a fricative and a vocalic segment, we select a token with a particularly long fricative, and the input has a particularly short fricative) Thus, it is very important that we select a token that is likely to match against input of any reasonable length and segmentation. In order to do this, we compute two important properties for each token: (1) the number of other tokens of the same identity that it can match, i.e., an alignment exists after applying all four constraint of the Hybrid Warp. (2) the difference between its length and the length of all tokens of this identity. The token that can match most other tokens is selected.[5] In case of a tie, we use the one closest to the average length.

---

[5]Because the thresholds in the Hybrid Warp were carefully chosen, most tokens can be warped against all other tokens in the same word.

Table III-1 shows the difference in recognition between the above method (as shown in Chapter 5) and averaging into the first token. The results show that this ordering process is helpful in selecting the most appropriate token to average into in the speaker independent case, but not for speaker-dependent case. This is because there are few, if any, "abnormal" tokens within one speaker's data, so choosing an arbitrary one is just as good.

Average into	Alphabets Speaker-Dependent	Speaker-Independent	Digits Speaker-Independent
Best Token	96.92%	65.77%	90.50%
First Token	96.92%	60.13%	87.00%

Figure III-1: Averaging training recognition by averaging into best or first token.

Two types of *Averaging Training* were implemented. The first is the non-discriminating type described in Chapter 5. The results in Chapter 5 and Table III-1 reflect averaging of this type. *All* tokens of the same word (except the very few cases where they cannot be aligned) are averaged together. This method is simple and fast; however, the final reference may be adversely affected by averaging abnormal tokens into it.

To add a discriminating factor in the averaging process, we introduce a second method. This discriminating averaging uses a threshold. All tokens of the same word are aligned and matched against each other. The largest group within which every distance is less than the threshold is averaged together to create the single reference template for that word. This may be thought of as a clustering technique with only one cluster.

A number of possible thresholds were tried for discriminating averaging. As the threshold is relaxed, more training tokens can be averaged. Table III-2 shows percentage recognized vs. percentage of tokens averaged into the final template.

Adding discrimination was not very helpful in the speaker-dependent case. It improved recognition by less than half percent, and the optimal recognition accuracy was obtained by averaging 95% of all tokens together. In the speaker-independent case, the improvement is almost 5%, and optimal accuracy is obtained by averaging 80% of all tokens together. This is intuitively correct because it is far more likely that there are tokens very different from each other in a speaker-independent database.

Approximate Percent Averaged	RECOGNITION % OF ALPHABETS Speaker-Dependent	Speaker-Independent
5%	91.21%	47.50%
10%	91.99%	48.08%
20%	92.21%	50.19%
35%	92.69%	55.00%
50%	93.22%	60.96%
70%	94.32%	67.50%
80%	94.62%	70.58%
85%	94.62%	70.00%
90%	95.64%	68.08%
95%	97.31%	65.96%
100%	96.92%	65.77%

Figure III-2: Percentage recognized vs. percentage of tokens used to create the final token.

III.4 Simplified Clustering

Simplified clustering is simply discriminating averaging that finds a cluster of only two tokens. A threshold is used to determine whether two tokens are "similar". The first two tokens found to be similar are averaged together to create the reference template for that word.

Initially, the first two tokens are tested. If their distance is under a threshold, they are averaged and the training stage for this word terminates. If their distance is above the threshold, the remaining tokens are introduced one by one. Each time, the current token is matched against all previous tokens, and the process terminates for the word as soon as the first match below the threshold is found. If all tokens in the training trial are exhausted without finding a pair, the first token is added as the reference.

Similar to the discriminating averaging, it is important to select a token to average into. Yet, unlike averaging where only one or few maximal size clusters can be found, simplified clustering may have many possible pairs with distance less than the threshold. In order to minimize the chance of averaging into a "bad token", all tokens are sorted[6] according to (1) how many other tokens of the same word they can match, and (2) how close they are to the average length for the word. Using a greedy algorithm, we process from the best to the worst, and when a pair is found, we always average the "worse" token into the "better" token. This effectively eliminates "unlikely" tokens. Table III-3 shows the difference between results obtained by first

[6] In averaging, sorting is not necessary. We only needed to find the best one in the cluster.

sorting the tokens according to goodness and those obtained by using random ordering, both using the optimal parameters for sorted ordering.

Ordering	Alphabets		Digits
	Speaker-Dependent	Speaker-Independent	Speaker-Independent
Sorted	95.38%	57.88%	82.00%
Unsorted	93.08%	50.77%	80.00%

Figure III-3: Simplified clustering recognition with and without sorting.

Again, a number of possible thresholds were tried. Table III-4 and III-5 show how percentage recognized is affected by percentage averaged after 2, 3, and 4 tokens have been considered for speaker independent and dependent recognition of alphabets.

Percent Averaged After Processing			Percent
2 tokens	3 tokens	4 tokens	Recognized
4%	8%	8%	43.08%
27%	50%	62%	47.50%
23%	50%	69%	44.42%
46%	73%	96%	45.58%
62%	85%	96%	47.31%
77%	96%	100%	50.77%
85%	100%	100%	55.77%
92%	100%	100%	54.81%
96%	100%	100%	55.77%
100%	100%	100%	57.88%

Figure III-4: Simplified clustering tuning for speaker independent alphabet recognition.

Surprisingly, for both speaker independent and dependent recognition, the best results were obtained by making the averaging threshold arbitrarily large, which always averages the first two tokens. This result differed from that of Rabiner [2], who used a threshold that allowed averaging of 62.1% of the first 2 tokens. The system in [2] is different from that described in a few minor places, but it is unlikely to result in such a significant difference.

We conjectured that the cause of this difference is the sorting process that took place. To confirm this hypothesis, the clustering algorithm was run on the same data without the "goodness sorting". The results are shown in Table III-6 and Table III-7.

Percent Averaged After Processing 2 tokens	3 tokens	4 tokens	Percent Recognized
30%	42%	42%	91.23%
50%	73%	73%	91.92%
77%	89%	93%	90.77%
96%	97%	100%	93.08%
96%	100%	100%	94.62%
100%	100%	100%	95.38%

Figure III-5: Simplified clustering tuning for speaker dependent alphabets.

Percent Averaged After Processing 2 tokens	3 tokens	4 tokens	Percent Recognized
0%	0%	0%	45.19%
10%	23%	23%	51.73%
45%	58%	77%	56.92%
60%	66%	82%	53.65%
77%	89%	100%	51.73%
92%	93%	100%	53.65%
92%	100%	100%	54.23%
97%	100%	100%	54.23%
100%	100%	100%	54.23%

Figure III-6: Simplified clustering tuning without sorting for speaker independent alphabet recognition.

Percent Averaged After Processing 2 tokens	3 tokens	4 tokens	Percent Recognized
12%	14%	23%	91.23%
40%	53%	73%	93.08%
72%	81%	89%	95.38%
96%	97%	100%	93.08%
100%	100%	100%	93.08%

Figure III-7: Simplified clustering tuning without sorting for speaker dependent alphabets.

This experiment produced results more similar to Rabiner's [2]. It also confirms our conjecture that the sorting process that took place is quite similar to the simplified clustering technique. Results in Table III-4

and III-5 show that when the tokens are sorted, we can simply match the first two without the simplified clustering process. Moreover, we see that given their optimal thresholds, sorted and unsorted ordering yield approximately the same results.

The sorting and clustering processes are similar in that they both try to find reliable tokens that can be averaged. The difference between them is that the sorting process has a more global view by examining the relationship between all pairs, but the information is less reliable because only alignability and duration are considered. The simplified clustering has a more local view by stopping when two tokens are found, but uses full distance calculation. Discriminating averaging in the previous section is a technique that uses global information with full distance calculation, and as expected, attained superior results. Nevertheless, results show that all single template techniques are limited to speaker dependent recognition.

III.5 K-Means Clustering

Section 5.5 explains this simple but effective clustering method in sufficient detail, so we will not reiterate here. Compared against other clustering techniques, K-Means Iteration has a number of advantages:

- It is free of thresholds, except for K (the number of clusters per word).
- There is no need of interactive supervision.
- It reliably finds K clusters.
- It is very simple.

Table III-8 shows recognition rate as a function of K. In the speaker dependent case where there are 5 training tokens for each word, recognition accuracy saturates when $K=3$. In speaker independent recognition where there are 20 training tokens for each word, $K=9$ is required to obtain the maximal accuracy.

One disadvantage of the K-Means Iteration is that it is not guaranteed to converge. It is possible for the method to oscillate between two or more cluster center configurations. This will occur if and only if a configuration is repeated. Therefore, we added a check for repeating configurations. In that case, the repeating configuration is used as the final cluster centers. This correction, of course, requires additional time. But it eliminates the need for supervision and intervention.

Levinson noted that generally convergence does occur [6]. Likewise, our experiments never resulted in repetitive configurations. Thus, we removed the check for repetition, and the time usage shown in Chapter 5 were obtained without the repetition check.

K (Number of Clusters)	Speaker-Dependent Alphabets	Speaker-Independent Alphabets
2	97.50%	77.88%
3	98.08%	79.81%
4	96.54%	82.31%
5	96.92%	82.50%
6		83.27%
7		84.62%
8		88.27%
9		90.19%
10		88.27%
11		88.27%
12		89.23%
13		89.23%
14		89.62%
16		88.65%

Figure III-8: K-Means recognition rate using different values for K.

Acknowledgments

The author wishes to thank D. R. Reddy for his motivation, suggestions, and guidance; K. Shikano for providing the LPC routines; R. Cole, A. Rudnicky, R. Stern, and A. Waibel for reading drafts of this paper; and G. Bradshaw for initiating this research effort.

References

1. Lowerre, B. T., *The HARPY Speech Recognition System*, PhD dissertation, Carnegie-Mellon University, April 1976.

2. Rabiner, L. R., Wilpon, J. G., "A Simplified, Robust Training Procedure for Speaker Trained, Isolated Word Recogniton Systems", *Journal, Acoustic Society of America*, Vol. 68, No. 4, November 1980, pp. 1271-1276.

3. Li, Z., Alleva, F., Reddy, R., "Effect of Reference Set Selection on Speaker Dependent Speech Recognition", Tech. report, Carnegie-Mellon University, July 1981.

4. Martin, T. B., "Practical Applications of Voice Input to Machine", *Proc., IEEE Transactions on Acoustics, Speech, Signal Processing*, Vol. 64, April 1976, pp. 487-501.

5. Rabiner, L. R., "On Creating Templates for Speaker Independent Recognition of Isolated Words", *IEEE Trans. on Acoustics, Speech, and Signal Processing*, Vol. ASSP-26, No. 1, February 1978, pp. 34-42.

6. Levinson, S. E., Rabiner, L. R., Rosenberg, A. E., Wilpon, J. G., "Interactive Clustering Techniques for Selecting Speaker-Independent Reference Templates for Isolated Word Recognition", *IEEE Trans. on Acoustics, Speech, and Signal Processing*, Vol. ASSP-27, No. 2, April 1979, pp. 134-41.

7. Gill, G., Goldberg, H., Reddy, R., Yegnanarayana, B., "A Recursive Segmentation Procedure for Continuous Speech", Tech. report, Carnegie-Mellon University, May 1978.

8. Bradshaw, G. L., *Learning to Understand Speech Sounds : A theory and model*, PhD dissertation, Carnegie-Mellon University, November 1983.

9. Kopec G., and Bush, M., "Evaluation of a Network-Based Isolated Digit Recognizer Using the TI Multi-Dialect Database", *Proceedings of the 1985 International Conference on Acoustics, Speech, and Signal Processing*, IEEE, April 1985, pp. 846-9.

10. Nocerino, N. S., "A Comparative Study of Distance Measures for Speech Recognition", Master's thesis, Massachusetts Institute of Technology, June 1985.

11. Sugiyama, M., Shikano, K, "LPC Peak Weighted Spectral Matching Measures", *Electronics and Communications in Japan*, Vol. 64-A, No. 51981, pp. 50-8.

12. Shikano, K, "Evaluation of LPC Spectral Matching Measures for Phonetic Unit Recognition", Tech. report, Carnegie-Mellon University, May 1985.

13. Itakura, F, "Minimum Prediction Residual Principle Applied to Speech Recognition", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. ASSP-261975, pp. 67-72.

14. White, Neeley, "Speech Recognition Experiments with Linear Prediction, Bandpass Filtering, and Dynamic Programming", *IEEE Trans. on Acoustics, Speech, and Signal Processing*, Vol. ASSP-24, April 1976, pp. 183-8.

15. Waibel, A., Yegnanarayana, B., "Comparative Study of Nonlinear Time Warping Techniques in Isolated Word Speech Recognition Systems", Tech. report, Carnegie-Mellon University, June 1981.

16. Waibel, A., Krishnan, N., Reddy, R., "Minimizing Computational Cost for Dynamic Programming Algorithms", Tech. report, Carnegie-Mellon University, June 1981.

17. Aho, A., Hopcroft, J., Ullman, J., *The Design and Analysis of Computer Algorithms*, Addison-Wesley Publishing Company, 1974.

18. Davis, S.B, P. Mermelstein, "Comparison of Parametric Representations of Monosyllabic Word Recognition in Continuously Spoken Sentences.", *Proc., IEEE Transactions on Acoustics, Speech, Signal Processing*, Vol. 28, August 1980, pp. 357-366.

19. Waibel, A., "Personal Communications", .

20. Bradshaw, G. L., Cole, R. A., Li, Z., "A Comparison of Learning Techniques in Speech Recognition", *Proceedings of the 1982 International Conference on Acoustics, Speech, and Signal Processing*, IEEE, May 1982, pp. 554-557.

21. Rabiner, L. R., Levinson, S. E., Rosenberg, A. E., Wilpon, J. G., "Speaker-Independent Recognition of Isolated Words Using Clustering Techniques", *IEEE Trans. on Acoustics, Speech, and Signal Processing*, Vol. ASSP-27, No. 4, August 1979, pp. 336-349.