Training Set Design for Connected Speech Recognition

Michael K. Brown, Maureen A. McGee, Lawrence R. Rabiner, *Fellow, IEEE,* and Jay G. Wilpon, *Senior Member, IEEE*

Abstract—**A key issue in the design and implementation of a speech recognition system is how to properly choose the speech material used to train the recognition algorithm. Training may be more formally defined as supervised learning of parameters of primitive speech patterns (templates, statistical models, etc.) used to characterize basic speech units (e.g., word or subword units), using labeled speech samples in the form of words and sentences.**

In this paper we study two methods for generating training sets. The first uses a nondeterministic statistical method to generate a uniform distribution of sentences from a finite state machine (FSM) represented in digraph form. The second method, a deterministic heuristic approach, takes into consideration the importance of word ordering to address the problem of coarticulation effects necessary for good training. The two methods are critically compared.

The first algorithm, referred to as MARKOV, converts the FSM into a first-order Markov model. The digraphs are determinized, transitive closure computed, transition probabilities are assigned, and stopping criteria established. An efficient algorithm for computing these parameters is described. Statistical tests are conducted to verify performance and demonstrate its utility. MARKOV is shown to be an excellent method for generating widely distributed coverage of the grammar, which includes a large number of distinct word bigrams for small samples, but performs less well as the sample sizes increase.

We then describe a second algorithm for generating training sentences, referred to as BIGRAM, that uses heuristics to satisfy the three requirements described below. It is well known that the highest recognition performance is achieved when the training set material is carefully selected to have the following properties: 1) adequate coverage of basic speech (subword) units; 2) adequate coverage of words in the recognition vocabulary (intraword contextual units); 3) adequate coverage of word bigrams (interword contextual units).

The first two conditions are necessary to ensure sufficient data for reliable and robust estimates of basic unit/word model parameters. The third condition is necessary to characterize the effects of word coarticulation on both unit and word models. BIGRAM uses criteria based on these requirements to generate an efficient (small) set of sentences that cover all word bigrams or, for incomplete training sets (e.g., with a large FSM where complete coverage is unattainable), a relatively large portion of word bigrams.

The training script generation algorithms are evaluated on two standard finite state grammars, the Flight Information and Reservation Language (FIRL) and the 991 word Darpa Naval Resource Management Grammar. BIGRAM compares favorably with MARKOV for generating larger training sets. A comparison of the BIGRAM generated training script for the Darpa task with a partially hand-processed training script obtained from BBN Laboratories shows that the automatic algorithm provides better coverage of the word bigrams in the language than the hand-generated script.

Manuscript received June 8, 1989; revised June 7, 1990.
The authors are with AT&T Bell Laboratories, Murray Hill, NJ 07974.
IEEE Log Number 9143804.

I. Introduction

PROPER training of speech recognizers is essential for obtaining the highest possible recognition accuracy for a given amount of speech material (e.g., Mikkilineni *et al.* [1]). With the introduction of the Grammar Compiler [2], rapid development of new grammars for connected speech recognition became possible. The need to rapidly generate good training sets from these grammars has led to the development of tools to train the recognizer with a sufficiently broad selection of the potential phrases. For small grammars having only a small number of possible sentences this is not difficult; simply use all possible sentences in the training set. However, the Grammar Compiler has made it easy to generate and use grammars capable of generating billions of sentences. It thus becomes necessary, for proper training, to carefully select some modest subset of all possible sentences.

Training of a word-based connected speech recognizer generally consists of a bootstrapping step, where the vocabulary words are trained using a data base consisting solely of isolated tokens of each word, followed by full training using a data base of fluently spoken sentences. Using the segmental k-means training procedure [3], the individual words in each sentence are extracted using a maximum likelihood segmentation scheme. Updated word reference models are generated and the process is iterated to convergence. This training procedure enables the recognition system to learn about and characterize word boundary coarticulation phenomena.

Two training set generation algorithms are studied in this paper. The first, subsequently referred to as MARKOV (developed by Brown and McGee), is a nondeterministic statistical method that converts an acyclic finite state machine (FSM) representation of the grammar into a first-order Markov model, and selects sentences in a uniformly distributed manner. The second method, referred to as BIGRAM (developed by Brown, Rabiner and Wilpon), is a deterministic, heuristic algorithm that attempts to cover all or, at least, most of the word bigrams in the grammar in a minimal number of sentences. Later, we compare the performance of the two methods on several standard tasks.

1053-587X/91/0600-1268$01.00 © 1991 IEEE

A. The MARKOV Algorithm

In our initial approach, uniform random selection of sentences, the intention is to produce a training set that covers the grammar as broadly as possible given a limited sample size. It should be noted that while it is always possible to obtain uniformly distributed sentences, in general, it is not possible to generate an arbitrary nonuniformly distributed set of sentences using the technique to be described when the FSM is efficiently represented because the structure of the digraph representation causes words in the grammar to be shared among various sentences. Thus, these sentences cannot be entirely independent. We will show that uniform sampling is an excellent method for obtaining broadly distributed coverage for small samples of large grammars, but the method does not guarantee that all words in the grammar will be represented. The underrepresentation can be corrected by a secondary algorithm like that described by Mikkilineni *et al.* [1]. Rather than augment the statistical method with a secondary algorithm, however, we develop this heuristic style of algorithm substantially further as another standalone method.

We show in Sections III-B and IV-A that, while selecting small training sets from large grammars at random is generally a good method of sparsely covering the full language uniformly including a high number of distinct word bigrams, selecting relatively large training sets from small grammars generally yields poor results. For strictly random selection, the correct solution is to use the *a priori* probabilities of sentence occurrence in actual practice to weight the sentence selection. Unfortunately, *a priori* sentence statistics are generally unavailable, and even if available, can often not be represented in an arbitrary Markov graph structure.

B. The BIGRAM Algorithm

Of particular importance in speech recognizer training are the word-pair (word bigram) coarticulation effects due to the slurring of one word into another. Word coarticulation is a normal characteristic of fluently spoken speech and becomes even more important when sentences are spoken rapidly. Recognition accuracy is improved with bigram-based training when compared with random training sets. Earlier evidence of this is found in Mikkilineni *et al.* [1]. Subsequent experiments (unpublished) have confirmed that bigram-based training results in significantly better recognition performance.

At the word level we do not need to be as concerned with word trigrams as with word bigrams since, with the exception of a few monosyllabic words, there is essentially no coarticulation across groups of more than two words. Hence, for effective training, we need only cover all possible word bigrams to generate essentially all possible word coarticulation phenomena. Note that if we produce a training set that contains all vocabulary words and all word bigrams, then we have covered virtually all of the training phenomena of interest.

It is clearly desirable, but not always possible, to cover all possible word bigram coarticulation effects for a given grammar. For small to moderate sized grammars, where the number of word bigrams in the language is less than about 1000, we can produce training sets with complete coverage. The average talker, however, will not want to train the recognizer (i.e., speak training sentences to the recognizer) with more than about 1000 sentences. For large grammars, where we cannot possibly cover all word bigrams, it is clearly desirable to cover as many word bigrams as possible in a reasonably small training set. Furthermore, since short sentences are easier to speak without interruption, and it may be important that all word bigrams in the sentence are trained (any pause in speaking causes the word bigram at that location to be lost), it is desirable to use as few words as possible in each training script sentence.

Previous efforts to produce effective training sets have used various methods. Some training sets have been produced by hand or by a combination of machine and hand selection [4]. Others have been produced automatically but do not cover all of the word bigrams in the language [1].

There are cases where training sets that do not cover all of the vocabulary words and word bigrams lead to recognition problems. For example, BBN Laboratories produced a set of 1600 training sentences for the Darpa speech recognition task [4] containing over 5000 word bigrams and covering 987 words of the 991 word vocabulary. Two copies of this sentence list were recorded for use in training various speaker independent speech recognition systems. Because four vocabulary words were missing from the training speech data base, whole word based recognizers cannot be fully trained using this training set. Furthermore, even though there were over 27 000 words in the training corpus, a large percentage of the 991 words occurred fewer than 4 times (i.e., too few to make a sufficient training contribution).

We describe a new heuristic algorithm that searches for word bigrams and attempts to produce a training set of sentences containing the largest number of new word bigrams in each new sentence. In this way, if there is no restriction on the number of sentences, then all word bigrams are generated in a small number of sentences. If the number of sentences must be limited, then a large set of unique word bigrams will be covered in the allowed sentences. We show that this fully automatic procedure produces training sentence sets with full vocabulary coverage and larger word bigram coverage than a semiautomatic method used by BBN to produce a training set for the Darpa grammar.

In the next section we discuss the MARKOV procedure for selecting sentences in a uniform way. Section III describes testing procedures used to verify that the training sets are indeed uniformly distributed. We study redundancy and language coverage in Section IV. In Section V the BIGRAM training set generation algorithm and its implementation is described. Real task grammars are tested

in Section VI along with an experimental evaluation and a comparison with previous training methods. Finally, we offer some concluding remarks in Section VII.

II. Grammatical Constraints

Speech recognizers generally use finite state machines (FSM's) to represent the grammatical constraints of a particular recognition task. Thus, the grammars are limited to regular grammars. Finite state or regular grammars are attractive for constraining speech recognizers because they place tight bounds on variability of speech and they are relatively easy to implement in a real-time processing system. These variability constraints limit the amount of acoustical processing needed to recognize a sentence, which is important for high processing speed. On the other hand, such grammars do not allow a very rich syntax for a given task unless the FSM is quite large. Undoubtedly, more sophisticated parsers such as augmented transition networks (ATN's) will eventually supercede when acoustical processing hardware becomes sufficiently fast.

The grammars, implemented as FSM's, are represented by digraphs. An example graph is shown in Fig. 1. Selecting sentences from such a grammar is not as simple as a uniform random walk traversing the graph from the start state to some arbitrary terminal state (indicated by double circles in the figures). Taking state transitions that have a large number of succeeding possibilities must be weighted more heavily. Ordinarily, deciding when to stop at an intermediate terminal state must be weighted based on both preceding possibilities and succeeding possibilities. We present a method for avoiding the intermediate stops that also has computational advantages.

A. Uniformly Distributed Grammar Sampling

The problem is to randomly select sentences from a very large set of possibilities, where the set of possibilities are contained in a FSM representation. The sentences must be selected with equal probability of selection so that no part of the language is underrepresented. It is easy to see from Fig. 1 that a random walk through the graph, where state transitions (or branches) are chosen with equal probability, will favor the selection of sentences "find an object" and "find another object." The sentences "find the object" and "find the next object" are only half as likely to occur because they must share the branch labeled "the," which is selected only as frequently as the branches labeled "an" and "another." Thus, the probabilities of occurrence of the sentences described by a random walk through the graph of Fig. 1 are:

1/3 find an object
1/3 find another object
1/6 find the object
1/6 find next object.

Clearly, the branch labeled "the" should be weighted twice as heavily as the other branches when selecting a branch at state 2. This weighting is given by the number

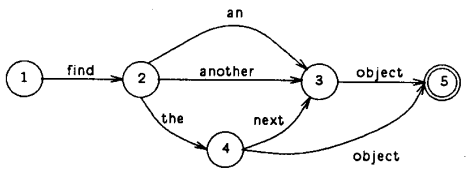


Fig. 1. State diagram for 'find (an | another | the (next | object & 1) object @ 1.'

of paths from the successor state 4 to all succeeding terminal states (in this trivial case only state 5).

The sentences may be selected more uniformly by creating a Markov model of the finite state grammar. State transition probabilities are determined from the relative numbers of sentence possibilities passing through each branch of the FSM. For efficiency reasons described later, we will not actually compute the full stochastic matrix of transition probabilities, but will instead store weights which can be computed directly from the structure of the graph.

For larger graphs the number of paths to terminal states is found by calculating the transitive closure on the graph. This can be obtained from repeated products of the connectivity matrix $\boldsymbol{C}$, which is defined as follows. For each pair of states (i, j) in the graph count the number of branches connecting state i to state j and enter this value in the $\boldsymbol{C}$ matrix at row i and column j. Thus, $\boldsymbol{C}$ indicates the number of ways of getting from state i to state j by element c_{ij} in one transition. The number of ways of getting from one state to another in two transitions is given by $\boldsymbol{C}^2$. Similarly, the number of ways of getting from state i to state j in n state transitions is given by element (i, j) of $\boldsymbol{C}^n$. For acyclic graphs $\boldsymbol{C}^n = 0$ for sufficiently large finite n.

The total number of all ways of getting from state i to another state j is given by element (i, j) of the transitive closure

$$\boldsymbol{B} = \sum_{i=0}^{\infty} \boldsymbol{C}^i = (\boldsymbol{I} - \boldsymbol{C})^{-1}. \tag{2.1}$$

Clearly, for very large graphs, direct computation of (2.1) is intractable. Typically, the graphs used for speech recognition purposes contain several hundred to several thousand states. Fortunately, we do not need all of the elements of $\boldsymbol{B}$ and there is an efficient way of finding the required elements.

1) Computing Transitive Closure Efficiently: In our application, the size of $\boldsymbol{C}$ is quite large, usually larger than 100×100. Our largest grammars have about 5000 states, so it becomes very important to consider storage and computation. The typical effort in computing the inverse of a matrix is $n^3/3$ flops. We can reduce the effort to $n(n-1)/2$ flops by modifying the structure of the matrix.

Since we only care about the number of ways of getting to terminal states we need compute only a few of the elements of $\boldsymbol{B}$. In general, the number of terminal states is

very small. Furthermore, $\boldsymbol{C}$ and hence $\boldsymbol{B}$ can be made triangular for acyclic graphs by ordering the states appropriately. This allows us to use an efficient algorithm for finding the weights. One additional trick that can be used is to add "null" branches to the graph from all intermediate terminal states to a single terminal state having no descendents (i.e., the state of the last column of the triangular $\boldsymbol{C}$ matrix). This modification does not alter the distribution of sentences selected because the transition probabilities take the new branches into account. The "null" branches are essentially cost free and have no labels so there is no alteration in the final training set results.

The ordering of states for an acyclic graph is obtained by a topological sort like that provided by the UNIX **tsort** command [5]. This guarantees that states are numbered so that a given state has a higher number than any of its ancestors. In fact, the **tsort** program source code was obtained and modified to perform this function in our implementation. Thus, making $\boldsymbol{C}$ upper triangular is straightforward.

The addition of the "null" branches increments by one all elements in the last column of $\boldsymbol{C}$ that fall in rows corresponding to terminal states, except for c_{nn}. This reduces the number of apparent terminal states to one. Now we only need to obtain solutions for the last column of $\boldsymbol{B}$, i.e., b_{in}, $i = 1, 2, \cdots, n-1$ (we do not actually need b_{1n} for training set generation but calculate it only to determine the number of sentences in the grammar). Since $(\boldsymbol{I} - \boldsymbol{C})$ is a unit diagonal upper triangular matrix, the inverse is easily found by reducing this matrix to $\boldsymbol{I}$ using Gauss elimination while applying the same operations to an augmentation matrix. In this case, however, that augmentation "matrix" is actually an $n \times 1$ column vector containing zeros except for the last element, which is one (i.e., the last column of $\boldsymbol{I}$). That is, we compute

$$b_{nn} = 1$$

$$b_{in} = \sum_{k=i+1}^{n} c_{ik} b_{kn}, \qquad i = n-1, n-2, \cdots, 1. \tag{2.2}$$

Since only the upper triangle needs processing it takes $n(n-1)/2$ flops to reduce (we reduce the matrix in reverse order one column at a time). We do not actually apply these operations to $(\boldsymbol{I} - \boldsymbol{C})$ because we do not care about its final form but instead apply them only to the augmentation vector. Thus, the number of flops required is $n(n-1)/2$.

One further observation reduces this effort to $O(n)$ in the number of adjacencies n_a (nonzero elements of $\boldsymbol{C}$). Since $\boldsymbol{C}$ is very sparse (usually no more than about 1% of the elements are non-zero) we need not compute all the terms of (2.2). Thus, a simple comparison for nonzero elements of $\boldsymbol{C}$ reduces the floating point computational effort to essentially n_a flops.

This algorithm is arguably optimal. Since closure is needed for every state the connectivity at each state must be examined at least once. This algorithm examines the connectivity exactly once.

The resulting b_{in} become the weights for all branches entering state i. Let the set of immediate descendents of state i be S_i. Then the stochastic transition probabilities are

$$p_{ij} = \frac{b_{jn}}{\sum_{k \in S_i} b_{kn}}.$$

Since we have merged all terminal states into one terminal state we no longer need to be concerned with stopping decisions at intermediate terminal states. We now have everything needed to start the Markov process.

2) Storage and Sentence Generation: Since we store the graph structure for the grammar in a doubly linked form, we do not need to actually store $\boldsymbol{C}$ but can instead compute the elements of $\boldsymbol{B}$ directly from the graph. The resulting weights are attached to the states rather than the branches and we look ahead one ply in the graph to compute the transition probabilities as the sentences are being generated. This may result in duplication of calculations but, as will be shown later, the amount of duplication is generally small and this method saves considerable storage since there are usually many more branches than states. Furthermore, many of the transition probabilities will never be computed for small training sets.

Then the procedure for generating one sentence is

1) Start at the start state, call this state i.
2) Order the branches leaving the state i.
3) Form a distribution of cumulative sums of probabilities p_{ij} for the ordered branches.
4) Generate a random number in the interval [0, 1).
5) Select the first ordered branch whose cumulative probability sum is greater than the random number.
6) Proceed to this next state (call it j) outputting the branch label if not "null."
7) If j is the terminal state,
   then stop,
   else go to 2.

B. Word Bigrams

Consider again the grammar depicted by the graph in Fig. 1. For this grammar the language consists of exactly four sentences:

1) find an object
2) find another object
3) find the next object
4) find the object.

and the grammar contains 5 states. There are 8 word bigrams in this language. They are:

1) find an
2) find another
3) find the

4) an object
5) another object
6) the next
7) the object
8) next object.

These word bigrams are completely represented in four training sentences (i.e., in this case all the sentences of this language). Using this set of four sentences we find that only one of the word bigrams ("find the") is represented more than once. For this example, the four sentences form the optimal training set in the sense that no smaller training set can be constructed that contains at least one occurrence of each word bigram.

In general, it is not possible to construct a training set, consisting of complete sentences, that contains all word bigrams in a grammar in a uniform way (i.e., equal number of occurrences of all word bigrams). The toy example given above illustrates why this is true. For larger, more realistic, grammars the nonuniformity of word bigram occurrence increases. It is clearly desirable to minimize this effect as much as possible so that individual word bigrams do not dominate the training set, thus causing some word bigrams to be poorly represented. This is accomplished by constructing training sentences by starting at a state in the grammar where the number of word bigrams previously generated in the training set is low and building a sentence from this point to the start state and to a terminal state in such a way that the number of new word bigrams generated is large. Searching a word bigram table for an appropriate starting place is undoubtedly the most important part of this process. The building phase can be made optimal by applying graph searching methods (and experiments were conducted with this method) but the processing time needed is considerable and the results are only marginally better than the heuristic approach to be described here. However, experiments have shown that using less effective starting word bigram heuristics (e.g., excluding some of the selection criteria) makes a much larger difference in the overall effectiveness of the algorithm.

III. Testing the Markov Training Set Generator

Two methods were used to test the statistical characteristics of the MARKOV training set algorithm. One test checks the large sample performance and the other tests the small sample performance. The large sample performance test shows that the distribution is truly uniform and the small sample test demonstrates that the samples are not spectrally band limited (i.e., locally correlated).

A. Distribution Test

To test the uniformity of the distribution, large training set samples were generated and counts taken of the frequency of occurrence for each possible sentence. To verify performance, three test grammars were used. These grammars were specifically designed to have characteristics typical of grammars commonly used in connected speech recognition and contain sufficient variability, including the intermediate terminal states, to make any bias in the selection procedure apparent. Indeed, several subtle problems were initially discovered by this procedure and subsequently corrected.

The grammars used for testing are shown in Figs. 2–4. These grammars differ only in the locations of terminal states. Note our convention of labeling the start state 0. In these figures state transitions have single letter labels. For simplicity, we have not shown multiple branches. Thus, multiple branches have multiple letter labels. For example, there are two branches (labeled B and C) from state 1 to state 2. The first grammar (G1) with 37 possible sentences has one intermediate terminal state at state 3. The second grammar (G2), having 41 sentence possibilities, has terminals at states 3 and 7. The third grammar (G3), containing 50 sentence possibilities, has three terminals at states 3, 4, and 7. The connectivity matrix for these grammars is

$$\boldsymbol{C} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 2 & 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Each test used a randomly generated training set of 100 000 sentences. If the generation of sentences is truly uniform, then for the grammars G1, G2, and G3 we expect a probability of occurrence p for each sentence generated to be $1/37$, $1/41$, and $1/50$, respectively. In an actual trial of 100 000 sentences we expect the distribution of measured frequencies to conform to the binomial distribution. Then the expectations in 100 000 sentences are that we will get about 2703 of each sentence for G1, about 2439 for G2, and about 2000 for G3. The expected standard deviations, given by $\sigma = (np(1 - p))^{1/2}$, where n is the number of Bernoulli trials, for grammars G1, G2, and G3 are 51.28, 48.78, and 44.27, respectively. We would expect normal deviates in any test to fall within 3σ of the expectations with a probability of 0.9973. Any outliers would indicate bias in the sentence generation.

In three tests consisting of 100 000 sentences each, the maximum deviates over the set of 37 for G1 were 2.24σ, 2.13σ, and 2.57σ. The average (rms) standard deviations on the full set of sentences, which are expected to be unity, were 1.09, 0.864, and 1.16. If we continued to repeat these trials there should be only about one chance in 370 that a deviate will exceed 3σ. The largest maximum deviate obtained over three trials each of 100 000

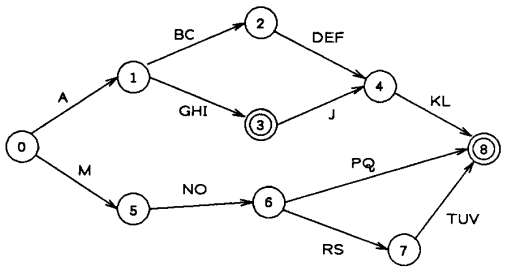


Fig. 2. Grammar G1.

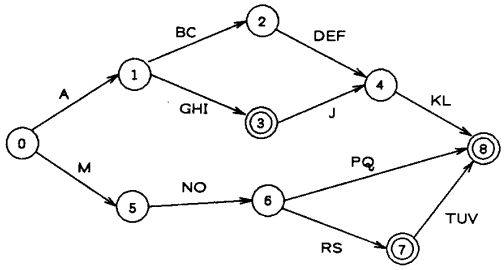


Fig. 3. Grammar G2.

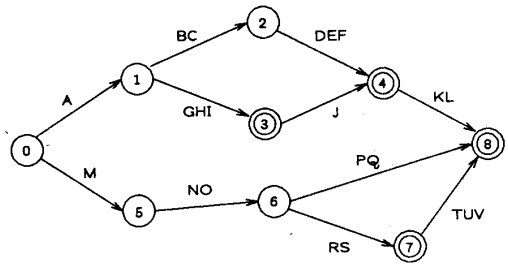


Fig. 4. Grammar G3.

sentence for G2 and G3 were 2.73σ and 2.30σ, respectively. The maximum deviations were consistently below 3σ from trial to trial. Thus it is apparent that the large sample statistics do indeed indicate uniformly distributed sentence selection.

One problem that was discovered during this testing phase was that the UNIX® C library function **rand()**, which uses a multiplicative congruential random number generator, is not adequate for generating the random numbers used in the state transition selection procedure. The function **random()**, which uses a nonlinear additive feedback random number generator, was subsequently used with much better results.

B. Occupancy Test

Since we are interested in creating small training sets from a large grammar it is important to test the small sample statistical performance. For very small samples some of the sentences in the grammar will not be present in the resulting training set, so the distribution test is not appropriate. The likelihood of missing sentences in a small training set can be found from the probability of occupancy in an n bin model, where n is the number of possible sentences.

One form of the occupancy model may be described as follows. Repeatedly throw balls at a set of n bins. Each ball must randomly fall into one of the bins. After r balls have been thrown we would like to know what the probability is that all of the bins are occupied by at least one ball. More specifically, we would like to determine the distribution of probabilities that exactly $1, 2, \cdots, n$ bins are empty. This is equivalent to randomly choosing one of n sentences from a set with replacement r times, each selection being independent of the others.

The probability that all of the bins are filled is given by [6]

$$p_0(r, n) = \sum_{\nu=0}^{n} (-1)^{\nu} \binom{n}{\nu} \left(1 - \frac{\nu}{n}\right)^{r} \tag{3.1}$$

and the probability that exactly m cells remain empty is

$$p_m(r, n) = \binom{n}{m} \left(1 - \frac{m}{n}\right)^{r} p_0(r, n - m). \tag{3.2}$$

Table I shows a numerical evaluation of (3.2).

One conclusion that can be drawn from (3.1) and (3.2) is that selecting samples that are only slightly larger than the number of sentence possibilities from the grammar at random is not a good method for obtaining a uniformly distributed sample since it is likely that some sentences will be repeated while others are ignored. We will deal with this issue more carefully in Section IV-A.

To test the small sample statistics we generated a large number of small independent training sets and compared the occupancy rates with the theoretical values obtained from (3.2). We looked for correspondence of the means and variance of the empirical distribution and the predicted distribution. In particular, if the number of occupied cells differs from expectation and the variance is significantly below expected variance we would be led to suspect that the random process is spectrally band limited, with potential for creating more redundancy than expected from an uncorrelated random selection.

For the first test we generated 300 training sets each containing 50 sentences for each of the grammars G1, G2, and G3. The resulting training sets were sorted and duplicate sentences eliminated. From (3.2), the predicted maximum likelihood sizes of the resulting training sets were about 28, 29, and 32 sentences, respectively. The expected values, which for relatively small r are generally about the same as the maximum likelihood values, are given by

$$\mu_m(r, n) = n\,[1 - (1 - n^{-1})^{r}] \tag{3.3}$$

which is derived from (3.2) using the well-known formula for expectation. For this experiment, the values computed using (3.3) are 27.6, 29.1, 31.8, respectively. The actual maximum likelihood estimates obtained from the training set data were about 28, 29.5, and 31.5, matching the pre-

TABLE I
PROBABILITIES $p_m(r, 5)$

m	$r = 5$	$r = 8$
0	0.03840	0.3225600
1	0.38400	0.5225472
2	0.48000	0.1483776
3	0.09600	0.0065024
4	0.00160	0.0000128

dicted values closely in all cases. The means of the distribution matched not only in location but showed good correspondence in magnitude as well. The variances were also quite similar.

A second test was also conducted using 300 training sets containing 25 sentences each. The expected sizes of the uniquely sorted training sets predicted from (3.3) were 18.3, 18.9, and 19.8 for sentences for G1, G2, and G3, respectively. The actual test estimates obtained on the uniquely sorted training sets were about 18, 19, and 20, respectively. The distributions also corresponded closely to (3.2).

One observation that is immediately apparent from the preceding analysis is that it is unlikely that any single random selection of r sentences from these small grammars will actually yield r different sentences. We observe that the number of sentence possibilities n has relatively little influence on the expected number of different sentences in the training set, particularly when n is relatively large. Later we show mathematical evidence for this conclusion. In each case for a selection of 50 sentences only about 30 different sentences were actually obtained. We study this characteristic of random selection more in the next section.

IV. Language Redundancy and Coverage

It is particularly important, for efficient training of a speech recognizer, that the training set cover the language of the grammar without a great deal of redundancy. Complete coverage of the language means that all vocabulary words must be present along with all possible word bigrams and trigrams, in order to model the coarticulation effects. For very large grammars complete coverage is clearly impractical. It is useful to study the likelihood of redundancy and coverage under the random selection rules we have presented.

A. Sentence Duplication

For training sets smaller than the number of sentence possibilities it is interesting to determine the probability p_d that a sentence will be duplicated since this is an undesirable event. It is particularly important to study these small sample properties because this is typical of the way that training sets will actually be generated in practice. It should be kept in mind that the following development assumes that the graphs have been minimized (using the methods described in Brown and Wilpon [2]) so that all redundancy has been removed. The probability of duplication can be found from (3.2) by summing over the possibilities that result in redundancy. That is,

$$p_d(r, n) = \sum_{m=n-r+1}^{n} p_m(r, n). \tag{4.1}$$

TABLE II
PROBABILITY $p_d(r, n)$ OF DUPLICATION

r n	10	100	1000	Estimate $\hat{p}_d(r, 1000)$
2	0.1000	0.0100	0.0010	0.001
3	0.2800	0.0298	0.0030	0.003
4	0.4960	0.0589	0.0060	0.006
5	0.6976	0.0966	0.0100	0.010
6	0.8488	0.1417	0.0149	0.015
7	0.9395	0.1932	0.0208	0.021
8	0.9819	0.2497	0.0277	0.028
9	0.9964	0.3097	0.0355	0.036
10	0.9996	0.3718	0.0441	0.045
20	1.0	0.8696	0.1741	0.190
30	1.0	0.9934	0.3625	0.435

Illustrative numerical values for (4.1) are given in Table II. It is clear from the table that even for very small training sets, the likelihood of duplication is surprisingly high when the grammar is of moderate size.

One way of increasing the number of different sentences obtained is to generate many sentences and eliminate the duplicates afterward. The size of the random sample needed to obtain n_s different sentences can be determined using (3.2). In this case we need an expression for the confidence we can have in obtaining at least α different sentences from a selection of r randomly chosen sentences, given n sentence possibilities. This is the probability that no more than $n - \alpha$ cells will remain empty after r trials

$$p(n_s \geq \alpha; r, n) = \sum_{m=0}^{\alpha-1} p_m(r, n)$$

which can be computed directly for moderate values of n, r, and α.

For very large n and relatively small r the likelihood of duplication is low, as can be seen from the first few entries in the second and third columns of Table II. For most speech recognition applications $r \ll n$ and duplication is not a serious problem. This can easily be seen from the following analysis.

From (3.3) we estimate the expected number of sentences given r trials using the approximation

$$(1 - n^{-1})^r \simeq 1 - \frac{r}{n}; \qquad r \ll n$$

which may be obtained from logarithmic approximations. Then the expected number of sentences

$$\mu_m(r, n) \simeq r.$$

Using this estimate we predict the probability of duplication on trial r is approximately $(r - 1)/n$. Then the

probability that a duplication will occur in the first r trials is approximately

$$p_d(r, n) \simeq \bar{p}_d(r, n) = \sum_{i=1}^{r} \frac{i-1}{n} = \frac{r(r-1)}{2n}.$$

It can be seen from Table II that this estimate is quite accurate for small r and large n.

Typical grammars being produced with the Grammar Compiler have values for n in excess of 10^{10}. In most cases it is not desirable to generate training sets larger than a few hundred sentences. For most of our recent applications $r^2 << n$ and the probability of duplication is very small indeed. Hence, for the very large grammars being produced today, we have an excellent method for choosing nonredundant training sets.

B. N-Gram Probabilities and Word Frequencies

The probability of making a particular state transition in any randomly selected sentence, say from state i to state j via branch k, is determined by the number of ways of reaching state i, the number of ways of terminating from state j, and the total number of possible sentences. That is,

$$p_k = \frac{b_{1i} b_{jn}}{b_{1n}} \tag{4.2}$$

where p_k indicates the probability of crossing branch k when randomly generating a single sentence as described earlier. Similarly, the N-gram probability of traversing any particular sequence of N branches from k through m starting at state s and ending at state t is

$$p_{k \cdots m} = \frac{b_{1s} b_{tn}}{b_{1n}}.$$

The expected frequency of occurrence of a vocabulary word in a randomly selected sentence is the sum of branch probabilities of all branches labeled with that word. That is,

$$f_w = \sum_{l(k)=w} p_k \tag{4.3}$$

over all k for word w, where $l(k)$ is the label of branch k.

Test grammars G1, G2, and G3 have word frequencies as shown in Table III for both equally probable sentence selection and for random selection based on equally probable branch selection at each state. For these grammars the word frequencies are also the word probabilities. These grammars are atypical in the sense that each branch has a unique label. In general, several branches will have the same word labels and the word frequencies may exceed unity, signifying that more than one instance of a word may occur in a single sentence.

Note that, on average, only one instance of the words "P" and "Q" will occur in 25 training sentences selected from grammar G3. Choosing sentences based on equally probable branch selection is only slightly better in this instance. The lowest representation in this case is for "T," "U," and "V" which require, on average, 16 sentences to get one instance while "A" only requires 2 sentences. Clearly, if it is desired to obtain a training set with equally balanced word frequencies, neither of these methods is appropriate.

TABLE III
WORD FREQUENCIES

	Equiprobable Sentences			Equiprobable Branches		
Word	G1	G2	G3	G1	G2	G3
A	0.567568	0.512195	0.6	0.5	0.5	0.5
B	0.162162	0.146341	0.18	0.1	0.1	0.1
C	0.162162	0.146341	0.18	0.1	0.1	0.1
D	0.108108	0.097561	0.12	0.0667	0.0667	0.0667
E	0.108108	0.097561	0.12	0.0667	0.0667	0.0667
F	0.108108	0.097561	0.12	0.0667	0.0667	0.0667
G	0.081081	0.073170	0.08	0.1	0.1	0.1
H	0.081081	0.073170	0.08	0.1	0.1	0.1
I	0.081081	0.073170	0.08	0.1	0.1	0.1
J	0.162162	0.146341	0.18	0.15	0.15	0.15
K	0.243243	0.219512	0.18	0.175	0.175	0.1167
L	0.233243	0.219512	0.18	0.175	0.175	0.1167
M	0.432432	0.487805	0.4	0.5	0.5	0.5
N	0.216216	0.243902	0.2	0.25	0.25	0.25
O	0.216216	0.243902	0.2	0.25	0.25	0.25
P	0.054054	0.048780	0.04	0.125	0.125	0.125
Q	0.054054	0.048780	0.04	0.125	0.125	0.125
R	0.162162	0.195122	0.16	0.125	0.125	0.125
S	0.162162	0.195122	0.16	0.125	0.125	0.125
T	0.108108	0.097561	0.08	0.0833	0.0625	0.0625
U	0.108108	0.097561	0.08	0.0833	0.0625	0.0625
V	0.108108	0.097561	0.08	0.0833	0.0625	0.0625

Generally, we do not want equally balanced word frequencies in the training sets because of coarticulation effects. Those words belonging to a large number of distinct word bigrams and trigrams need more representation in the training set. We discuss this further in the next section.

V. THE BIGRAM-BASED TRAINING ALGORITHM

The BIGRAM algorithm for constructing the training set consists of two phases. The first phase involves searching a table of identified word bigrams derived from the given grammar, and choosing a word bigram and its corresponding location in the grammar digraph as a starting point. The second phase consists of two parts: constructing the leading part of the sentence up to the starting word bigram location, and constructing the remainder of the sentence. Both of these parts are essentially the same process, but in the former we are looking for the start state in the grammar digraph, and in the latter we are looking for some terminal state in the digraph.

A. Selecting the Starting Bigram

We have found empirically that selecting a starting location in the grammar for generating a sentence is critical to obtaining high word bigram coverage for large grammars with a small number of training sentences. The selection process is based on a number of criteria that measure the likelihood of generating a sentence containing a

large number of new word bigrams. These criteria, in order of importance, are

1) the number of times this word bigram has already appeared;
2) the number of times the words in this word bigram have appeared;
3) the number of potential sentences that can be constructed from this starting location; and
4) how far this starting location is from the start state.

The grammar digraph is first preprocessed to determine which word bigrams are present and a table of word bigrams is constructed. The number of potential sentences is obtained from a transitive closure computation for the digraph.

For obvious reasons, the dominant selection criterion is the number of times a word bigram has already appeared. For very large grammars and small training sets this means searching for a word bigram that has not yet appeared. For smaller grammars it is desirable to have several copies of each word bigram, thereby attempting to distribute the coverage uniformly.

The next criterion is the number of times a word has already appeared. It is important to avoid using the same word many times in each of its possible word bigrams at a particular location in the grammar. Of course, each word in the vocabulary will appear at least once if all of the word bigrams are present. When all of the word bigrams are not present, as in the case of small training sets and large grammars, we have found that the number of new word bigrams generated by using the same word repeatedly is small. This is due to the fact that word bigrams tend to be highly correlated in some grammars. That is, if a sentence starts with a particular word pair (first word bigram) then the likelihood is high that the second word bigram will be one of a relatively small set of possible word bigrams. The starting word bigrams are selected based first on the lowest frequency of occurrence of the first word of the word bigram and then on the frequency of occurrence of the second word of the word bigram, although this ordering is arbitrary.

If all of the above criteria are met equally for more than one word bigram, we choose that word bigram contained in the fewest number of sentences of the language. The reason for this is somewhat subtle. In the sentence building part of the algorithm we will, all other things being equal, choose to take the path that gives us the greatest number of sentence choices (based on the argument that maximum freedom of choice allows us to more easily avoid repeating word bigrams). On the other hand, locations in the grammar that are traversed in only a small number of ways are less likely to be included in sentences that originate from other parts of the digraph. Since we would like to obtain as broad a coverage of the grammar as possible, we choose these unlikely locations as starting places.

We choose to start at a later state in the grammar if all of the other criteria are met equally since this compensates for the fact that later parts of the grammar will generally be avoided by the sentence building part of the algorithm due to its tendency to generate short sentences whenever possible.

B. Building the Remainder of the Sentence

Once a starting location has been chosen, the leading part of the sentence is constructed by searching the grammar digraph and the word bigram table simultaneously to minimize a path cost consisting of the following, in order of importance:

1) the number of times this word bigram has already appeared;
2) the minimum number of word bigrams over the set of possibilities at the immediate ancestor state;
3) the number of states (words) to the start state;
4) the number of potential sentences available at the ancestor state.

The first criterion is again obvious. The second criterion amounts to a one-ply search for the minimum word bigram cost path to the start state. We will discuss this further in the next subsection.

The third criterion is based on the desirability of short sentences for convenience in training. We look for the shortest path to the start state. The shortest path information is obtained from a preprocessing stage where dynamic programming is applied to the entire grammar digraph. Thus each state has associated with it the minimum path length and the direction of this path. Short sentences have the added benefit of reducing the correlation of word bigrams in the training set. That is, since word bigrams tend to be correlated within a sentence, there is more freedom to choose the word bigrams independently when the sentence length is short (and we have more sentences). This means that the number of duplicate word bigrams is reduced in the final training set.

The last criterion tries to maintain maximum entropy. All other criteria being equal, we wish to retain as much freedom of choice as possible as the sentence is being constructed so that we have a better chance of avoiding word bigrams that have already been used.

The construction of the remainder of the sentence, from the starting word bigram to a terminal state in the grammar digraph, is similar to the method just described. Of course, the direction of search in the digraph is reversed. The only other difference is in the path length criterion. Again, dynamic programming is applied in a preprocessing stage to determine minimum path length to the nearest terminal state. Thus, sentences that would normally be constructed by passing through a terminal state to a later terminal state will not be generated unless the starting word bigram is at a location later than a terminal state. That is, terminal states are unimportant when constructing the leading part of the sentence but we always choose to stop when we encounter a terminal state while constructing the trailing part of the sentence.

C. Optimal Sentence Building

Obtaining the globally minimal training set necessary to completely cover all word bigrams in the grammar is a difficult problem. It is easy to show that choosing a new starting word bigram that occurs in only one place (uniquely) in the grammar gives a starting location that must fall within a sentence of the minimal set (since this word bigram is the only one that can complete the set). Unfortunately, there is no simple way to guarantee that the rest of the sentence generated will be a member of the minimal set, even if all of the word bigrams for this sentence might have the uniqueness properties just mentioned. Global information is needed to make this determination which involves generating all possible sets for comparison. Clearly, this is prohibitively expensive, so we choose to apply heuristic methods to obtain a good but not minimal training set.

As mentioned in the preceding subsection, the second criterion requires looking one ply ahead to find the word bigram that will result in a path containing a larger total number of new word bigrams. The absolute maximum new word bigram count can be obtained by applying graph searching techniques. Unfortunately, for large grammar digraphs such graph searching methods are very time consuming. For experimental purposes a full search was tested. The results did not significantly improve for small to moderate sized grammars where the entire word bigram set was obtainable. For very large grammars the amount of time needed to generate sentences is prohibitive and no conclusion about effectiveness can be made at this time.

The single ply search does provide significant improvement over no search, and it is likely that a full search will provide additional improvement in new word bigram coverage when the training set covers a substantial part but not all of the possible word bigram set. It is not until many of the word bigrams have been used that the searching becomes necessary to avoid repeating word bigrams. When the training set is very small, in relation to the size of the grammar, the likelihood of repeating word bigrams in the sparsely covered word bigram set is small.

D. A Small Toy Grammar

The algorithm was first run on the small toy grammar shown in Fig. 2. The grammar consists of 9 states, has a vocabulary of 22 words, and the language contains 37 sentences. There are 38 word bigrams in this grammar and the algorithm covers all of them in 20 sentences. Of most interest, for a small grammar like this, is the number of sentences needed to cover each word bigram at least N times. The number of sentences needed for this task is given in Table IV.

It can be seen from Table IV that the heuristic BIGRAM algorithm, although not optimal, does very well. Interestingly, if we ignore the second criterion of Section V-A for starting word bigrams, then we obtain a minimal number of training sentences for this grammar. Unfortunately, this limited set of criteria results in poor performance on larger grammars. Since the small grammars require so few training sentences, even when the selection is not optimal, we choose to use the criteria best suited for the larger grammars.

TABLE IV
TRAINING SET SIZE

# Bigram Copies	Number of Sentences	Minimal Number
1	20	19
2	39	38
3	59	57
4	78	76
5	97	95

VI. EXPERIMENTAL EVALUATION

We tested the algorithm on two real grammars. These are representative of the typical grammars being used today. The flight information and reservation (FIRL) grammar [7], [8] is a relatively small hand-built grammar that was designed in the late 1970's. We are using a new version of that grammar (that we call new FIRL or NFIRL) that was constructed using the Grammar Compiler [2]. The DARPA [4] grammar, which is also constructed with the Grammar Compiler, is a large ship-based naval command grammar. The grammar statistics are shown in Table V. The number of adjacencies is the number of nonzero elements of the connectivity matrix.

The criteria used to measure the effectiveness of the algorithms include:

1) the number of distinct word bigrams contained in a fixed number of training sentences;
2) the number of distinct word bigrams per word in a training set;
3) the number of vocabulary words used in a fixed number of training sentences;
4) the number of sentences needed to completely cover the word bigram set.

It is desirable to minimize the last measure and maximize the other three. The first two are alternate measures of efficiency, depending on whether the number of sentences or number of words in the training set is of most importance. The training algorithms are written to generate training sets consisting entirely of complete sentences, and the number of sentences generated can be controlled. From the human trainer's point of view, the number of words probably more accurately represents the amount of speech he will need to input during the training session.

The results (see Tables VI and X) show that MARKOV performs better than BIGRAM for extracting small training sets from large grammars, but the performance of BIGRAM soon overtakes MARKOV as the training set size increases. During the testing of MARKOV, the number of redundant sentences generated was constantly monitored. No sentence redundancy was seen in any of the training sets, even for 1000 sentence sets generated from the NFIRL grammar.

TABLE V
REAL GRAMMAR STATISTICS

Grammar ☞	NFIRL	DARPA†
States	123	4686
Terminals	8	57
Branches	439	38329
Adjacencies	184	8392
Sentences	6.01×10^9	3.14×10^{14}
Bigrams	674	56918

†Optimized

TABLE VI
PERFORMANCE STATISTICS ON NFIRL

	Markov			BIGRAM		
# Sentences	# B	# V	# B/# W	# B	# V	# B/# W
10	106	44	0.612	30	31	0.555
50	267	68	0.304	237	119	0.592
100	329	76	0.188	351	127	0.411
295	416	83	0.080	674	127	0.242

The computation times for the two grammars were typically about 2.35 s for the NFIRL grammar and about 44.5 s for the DARPA grammar on a Sun 4/280 (a 10 mips machine). Considering the sizes of the grammars being processed, these running times are excellent.

A. The NFIRL Grammar

The NFIRL grammar has been used for more than a decade in connected speech recognition experiments. It is connected to an on-line airline flight schedule data base and can be used to inquire about and set up flight reservations. The vocabulary can be seen in Table VII.

Several training sets of sizes varying from 10 sentences to 295 sentences were generated and examined for word bigram and vocabulary coverage as described above. The results for both algorithms are shown in Table VI. BIGRAM produced complete word bigram and vocabulary coverage in 295 sentences. Since MARKOV is nondeterministic, the results vary from trial to trial, but the variance is small. The results in Table VI (and also Table X) for MARKOV are averaged over several trials. The columns of Table VI (and Table X) are labeled with the number of distinct word bigrams generated (#B), the number of vocabulary words included (#V), and the ratio of distinct word bigrams to total words in the training set (#B/#W).

At first glance, it appears that MARKOV significantly outperforms BIGRAM for small training sets (e.g., 106 word bigrams for MARKOV compared to only 30 word bigrams for BIGRAM in the 10 sentence sets). However, MARKOV does not have any constraint on the length of a sentence, so the sentences generated by MARKOV are about three times as long as those produced by BIGRAM. Even so, the ratios of word bigrams per training word still indicates that MARKOV does very well.

As the number of sentences in the training set increases, it becomes increasingly likely that previously generated word bigrams will appear again, and MARKOV starts to show performance loss. BIGRAM also loses efficiency, as indicated by the word bigram to word ratios, but it does not lose ground as quickly as MARKOV and soon yields much better performance.

One difficulty that MARKOV will have on some grammars is obtaining complete coverage of the vocabulary. The word frequencies for the entire NFIRL grammar are given in Table VII. These numbers indicate the expected number of occurrences of a word per sentence. For example, in a training set of 1000 randomly selected sentences we would expect to see about 463 instances of the word "my." This grammar has a 127 word vocabulary. Eleven of these words are used only once in the entire grammar. It is obvious from the extremely low word frequencies of many of the words that a random selection of 1000 sentences will, on average, not cause even one occurrence. At the other extreme, in a training set of 1000 sentences there will probably be more than 1000 instances of the word "on."

B. The Darpa Naval Resource Management Grammar

The Darpa grammar [4] is a large Naval Resource Management grammar developed by a group consisting of BBN Laboratories, Texas Instruments, Stanford Research Institute (SRI), and the National Bureau of Standards (NBS, now called the National Institute for Science and Technology or NIST). This is generally considered to be a good test grammar, and difficult recognition task. The original grammar, as distributed by BBN, contains over 60 000 states and more than 250 000 state transitions. As such it is essentially unusable with current technology for speech recognition purposes because of its size. The grammar was reduced (see Table V) using the optimization algorithm available with the Grammar Compiler [2]. The reduced grammar contains only about 15% of the original branches making it possible to process this large grammar for speech recognition purposes. The Darpa grammar comes with a training set of 3200 sentences that were generated by a combination of machine and hand processing. The 3200 sentences are actually two copies of a set of 1600 different sentences that contain 5109 word bigrams out of the 56 918 word bigrams that exist in the Darpa grammar.

To compare the performance of BIGRAM on large grammars with the BBN hand-generated training set, we generated a set of 1600 sentences from this grammar. The resulting training set statistics are shown in Table VIII. The statistics for the BBN training set, which included hand processing, is also shown for comparison. Clearly, BIGRAM produces (automatically) a smaller training set containing not only more word bigrams but also complete coverage of the vocabulary. Full vocabulary coverage is particularly important, as mentioned in Section I, because missing triphones in the training data can lead to problems in characterizing the basic speech units for recognition.

TABLE VII
WORD FREQUENCIES IN NFIRL

Word	Freq.	Word	Freq.	Word	Freq.
my	0.4633	the	0.5917	card†	1.664×10^{-10}
home	0.1544	morning	0.1073	prefer	2.496×10^{-9}
office	0.1544	night	0.1073	boeing	4.993×10^{-10}
phone	0.4633	afternoon	0.1073	douglas	4.993×10^{-10}
number	0.7284	evening	0.1073	dc	9.986×10^{-10}
is	0.4751	of	0.05364	ten	4.827×10^{-9}
area	0.4602	sunday	0.06708	bac†	1.664×10^{-10}
code	0.4602	monday	0.06708	lockheed†	1.664×10^{-10}
two	0.8311	tuesday	0.06708	eleven	4.327×10^{-9}
three	0.6931	wednesday	0.06708	how	0.07727
four	0.6586	thursday	0.06708	much	0.0002484
five	0.6586	friday	0.06708	many	0.007478
six	0.6586	saturday	0.06708	go	0.001048
seven	0.6586	january	0.04458	are	0.01148
eight	0.6586	february	0.04458	there	0.01148
nine	0.6586	march	0.04458	take	0.005491
zero	0.77	april	0.04458	plane	0.005491
one	0.9051	may	0.04458	a	0.05491
when	0.0002499	june	0.04458	meal	0.005491
at	0.00025	july	0.04458	served	0.005491
what	0.01173	august	0.04458	stops	0.01098
time	0.0004984	september	0.04458	want	0.1706
does	2.929×10^{-6}	october	0.04458	need	0.1647
flight	0.5328	november	0.04458	would	0.1706
to	0.5358	december	0.04458	like	0.1706
from	0.002883	please	2.164×10^{-9}	coach	0.1647
washington	0.05404	repeat	1.498×10^{-9}	first	0.1647
chicago	0.05404	times	4.993×10^{-10}	class	0.1647
new-york	0.05404	arrival	3.329×10^{-10}	seat	0.09883
boston	0.05404	departure	3.329×10^{-10}	seats	0.3953
denver	0.05404	fare	0.0004969	non-stop	3.329×10^{-10}
detroit	0.05404	i	0.5115	some	6.657×10^{-10}
los-angeles	0.05404	will	0.005491	information	6.657×10^{-10}
miami	0.05404	pay	8.322×10^{-10}	make	6.657×10^{-10}
philadelphia	0.05404	by	8.322×10^{-10}	reservation	6.657×10^{-10}
seattle	0.05404	cash†	1.664×10^{-10}	twelve	3.994×10^{-9}
depart	0.0001058	american†	1.664×10^{-10}	o-clock	1.598×10^{-8}
arrive	1.465×10^{-6}	express†	1.664×10^{-10}	am	1.598×10^{-8}
do	0.0004969	diners†	1.664×10^{-10}	pm	1.598×10^{-8}
flights	0.002484	club†	1.664×10^{-10}	in	5.326×10^{-9}
leave	0.001595	master†	1.664×10^{-10}	return	0.0001044
for	0.00159	charge†	1.664×10^{-10}		
on	1.053	credit†	1.664×10^{-10}		

†Only one occurrence in NFIRL

TABLE VIII
DARPA-1000 STATISTICS

Statistic	BIGRAM	BBN
# Sentences	1600	1600
# Words Included	991	987
# Bigrams Included	5496	5109
Total Number of Words	11771	13987
Max. Sentence Length	20	22
Ave. Sentence Length	7.36	8.74

TABLE IX
RANDOM SENTENCE STATISTICS

Test	# B	# V
1	2081	252
2	2087	243
3	2052	246

This phenomenon was observed when using the BBN training set with this grammar in a phone-like unit (PLU) based recognizer.

For comparison purposes a random selection of 1600 sentences from the Darpa grammar was made using MARKOV. The number of word bigrams and vocabulary words covered by three random samples of 1600 sentences each are given in Table IX. From these results it is clear that random sentence selection is not an effective way of getting high word bigram coverage and the technique is inadequate for obtaining good vocabulary coverage. As seen previously with NFIRL, the poor vocabulary coverage is due to very low word monogram frequencies for some words in the grammar.

The complete statistics for 1600 sentences and several smaller training sets are summarized in Table X. Again, MARKOV does well for small training sets, but soon loses to BIGRAM as the set size increases. For this grammar, BIGRAM has completely covered the vocabulary in 500

TABLE X
PERFORMANCE STATISTICS ON DARPA

	Markov			BIGRAM		
# Sentences	# B	# V	# B/# W	# B	# V	# B/# W
10	152	88	0.695	23	32	0.697
100	650	159	0.299	367	309	0.734
500	1387	219	0.128	2343	991	0.651
1600	2073	247	0.060	5496	991	0.467

sentences, while MARKOV is still far from complete coverage even at 1600 sentences.

VII. Concluding Remarks

Two new methods, called MARKOV and BIGRAM, for generating training sentence sets have been developed. MARKOV is a method for generating small uniformly distributed speech recognition training sets. Using this procedure we can create training subsets that will give broad coverage of the language for a finite state grammar when the size of the training set is much smaller than the number of sentence possibilities. However, we have also shown that, as the training set size increases, individual word coverage may be very poor, depending on the frequency of word occurrences in the grammar. Random sentence selection can be useful for generating small training sets quickly since MARKOV is a much faster algorithm than BIGRAM and, in many cases, may be sufficient for final training.

Uniformly distributed training sets are sometimes not the best choice for good recognition accuracy. In particular, if the grammar is of modest size, containing perhaps only a few thousand sentence possibilities, then a randomly chosen training set of a few hundred sentences will probably be biased. For these cases, we have derived useful estimates for the probability of sentence duplication. Since most of our grammars are now very large relative to the training sets, this may not be a problem. However, one important figure of merit, word bigram coverage, may be quite poor when using this method.

To address this problem we have also developed a heuristic method (BIGRAM) for generating training sets having excellent coverage of both word bigrams and vocabulary for a given finite state grammar. The results are shown to be favorable when compared to simple random sentence selection for relatively large training sets.

These new tools complement the other tools developed in conjunction with the Grammar Compiler [2]. Experiments have shown that speech recognizers trained using these techniques will demonstrate improved performance over those trained from more conventional scripts.

References

[1] R. P. Mikkilineni, J. G. Wilpon, and L. P. Rabiner, "A procedure to generate training sequences for a connected word recognizer using the segmental *k*-means training algorithm," in *Proc. IEEE ICASSP'88*, vol. S, 1988, pp. 433-436.

[2] M. K. Brown and J. G. Wilpon, "A grammar compiler for connected speech recognition," *IEEE Trans. Signal Processing*, vol. 39, no. 1, pp. 17-28, Jan. 1991.

[3] L. R. Rabiner, J. G. Wilpon, and B. H. Juang, "A segmental *k*-means training procedure for connected word recognition," *AT&T Tech. J.*, vol. 65, no. 3, pp. 21-31, May-June 1986.

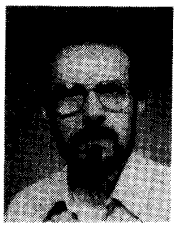
[4] P. Price, W. M. Fisher, J. Bernstein, and D. S. Pallet, "The Darpa 1000-word resource management database for continuous speech recognition," in *Proc. IEEE ICASSP'88*, 1988.

[5] *UNIX® Time-Sharing System, Programmers Manual*, vol. 1, ninth ed., AT&T Bell Laboratories, Sept. 1986.

[6] W. Feller, *An Introduction to Probability Theory and Its Applications*, vol. 1, 3rd ed. New York: Wiley, 1968.

[7] S. E. Levinson and L. R. Rabiner, "A task-oriented conversational mode speech understanding system," in *Speech and Speaker Recognition*, M. R. Schroeder, Ed. Basil, Switzerland: S. Karger AG, 1985, pp. 149-196.

[8] M. M. Sondhi and S. E. Levinson, "Computing relative redundancy to measure grammatical constraint in speech recognition tasks," in *Proc. IEEE ICASSP'78*, 1978.



Michael K. Brown received the B.S. degree in 1973, the M.S. degree in 1977, and the Ph.D. degree in 1981, all from the University of Michigan, Ann Arbor, in electrical engineering.

From 1973 to 1976 he was with the Burroughs Corporation (now Unisys Corporation) where he was involved in the development of ink jet printing systems. From 1976 to 1980 he continued with Burroughs as a consultant working on unconstrained handwritten character recognition while pursuing the Ph.D. degree at the University of Michigan. His dissertation, on the topic of cursive script recognition, described new techniques in feature extraction and pattern recognition. In 1980 he joined the Speech Processing Group at AT&T Bell Laboratories, Murray Hill, NJ, where he was involved in the development of speech recognition algorithms and VLSI hardware. Since 1983 he has been with the Interactive Systems Research Department (formerly called Robotics Principles Research Department) pursuing interests in electromechanical control, sensory perception, and man-machine interaction. He holds 6 patents in the area of control systems and speech processing, and has written extensively on speech, control, sensors, and robotics.

Maureen A. McGee received the B.S. degree in computer science from Monmouth College, West Long Branch, NJ, and the M.S. degree in computer science from New York University in 1986.

She has been a Member of the Technical Staff at AT&T Bell Laboratories, Murray Hill, NJ, since 1986. Her present areas of research include both speaker-dependent and speaker-independent connected work speech recognition. Over the previous 12 years, she has been involved in a number of diverse research and development projects, including process control simulation, operating systems, and compiler generation.

Lawrence R. Rabiner (S'62–M'67–SM'75–F'75) was born in Brooklyn, NY, on September 28, 1943. He received the S.B. and S.M. degrees simultaneously in June 1964, and the Ph.D. degree in electrical engineering in June 1967, all from the Massachusetts Institute of Technology, Cambridge.

From 1962 through 1964 he participated in the cooperative plan in electrical engineering at Bell Laboratories, Whippany and Murray Hill, NJ. He worked on digital circuitry, military communications problems, and problems in binaural hearing. Presently he is engaged in digital signal processing techniques at Bell Laboratories, Murray Hill. He is coauthor of the books *Theory and Application of Digital Signal Processing* (Prentice-Hall, 1975), *Digital Processing of Speech Signals* (Prentice-Hall, 1978), and *Multirate Digital Signal Processing* (Prentice-Hall, 1983).

Dr. Rabiner is a member of Eta Kappa Nu, Sigma Xi, Tau Beta Pi, the National Academy of Engineering, the National Academy of Sciences, and is a Fellow of the Acoustical Society of America, and AT&T Bell Laboratories.

Jay G. Wilpon (M'84–SM'87) was born in Newark, NJ, on February 28, 1955. He received the B.S. and A.B. degrees (*cum laude*) in mathematics and economics, respectively, from Lafayette College, Easton, PA, in 1977, and the M.S. degree in electrical engineering/computer science from Stevens Institute of Technology, Hoboken, NJ, in 1982.

Since June 1977 he has been with the Speech Research Department at AT&T Bell Laboratories, Murray Hill, NJ, where he is a Member of the Technical Staff. He has been engaged in speech communications research and is presently concentrating on problems in isolated and connected word speech recognition. He has published extensively in this field and has been awarded several patents. His current interests lie in training procedures for both speaker dependent and speaker independent recognition systems, keyword spotting algorithms, speech detection algorithms, and determining the viability of implementing speech recognition systems for general usage over the telephone network.

Mr. Wilpon received in 1987 the IEEE Acoustics, Speech, and Signal Processing Society's Paper Award for his work on clustering algorithms for use in training automatic speech recognition systems.