Nonparametric and Linguistic Approaches to Pattern Recognition

philip h. swain
and
king-sun fu

Laboratory for
Applications of
Remote
Sensing

Purdue University

LARS Information Note 051970

NONPARAMETRIC AND LINGUISTIC APPROACHES TO PATTERN RECOGNITION

P. H. Swain
K. S. Fu

TR-EE 70-20

June, 1970

School of Electrical Engineering
Purdue University
Lafayette, Indiana 47907

This work was supported by National Science Foundation Grant GK-1970; AFOSR Grant 69-1776; United States Department of Agriculture, Agricultural Research Service Contract 12-14-100-1029(20); and National Aeronautics and Space Administration Research Grant NGL 15-005-112.

TABLE OF CONTENTS

TABLE OF CONTENTS, cont.

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES, cont.

ABSTRACT

This report investigates two approaches to pattern recognition which utilize information about pattern organization. First, a nonparametric method is developed for estimating the probability density functions associated with the pattern classes. The dispersion of the patterns in the feature space is used in attempting to optimize the estimate. The second approach involves the structural relationships of pattern components, an approach called "linguistic" because it employs the concepts and methods of formal linguistics.

The nonparametric density estimation technique is shown to produce acceptable results with real data and demonstrates a definite advantage over a parametric procedure when multimodal data is involved.

Two alternative techniques are investigated for analyzing linguistic descriptions of patterns. Stochastic automata are considered as recognizers of stochastic pattern languages, and it is shown that of the types of recognition criteria which have been defined for automata to date, recognition with zero cut-point is probably the most suitable for recognizing infinite stochastic pattern languages (languages consisting of an infinite number of

patterns). A stochastic generalization of the recently proposed programmed grammar is developed and an algorithm is implemented which analyzes the languages described by stochastic programmed grammars. A grammar is evolved which generates a language of "noisy" two-dimensional patterns, and the analysis algorithm is applied to these patterns to demonstrate its performance.

CHAPTER 1

INTRODUCTION

This report investigates two approaches to pattern recognition which utilize information about pattern organization. The first approach has much of the flavor of "classical" statistical pattern recognition in that it characterizes the pattern classes in terms of their associated probability density functions. To do this effectively, use is made of some relatively simple relationships among patterns and groups of patterns in order to improve a nonparametric estimation of the density functions. The second approach represents a fairly radical departure from classical pattern recognition. It concerns the representation and manipulation of structural relationships within patterns when such relationships are essential distinguishing features. The approach is termed linguistic because it utilizes the philosophy and methods of formal linguistics in developing grammars which characterize patterns.

1.1 Nonparametric Statistical Pattern Recognition

The characteristics of many practical pattern recognition problems suggest the use of statistical techniques for solution of these problems. Such characteristics include, for example:

1. Incidental feature variations which tend to obscure differences between pattern classes (noisy data).

2. Uncertainty, however small, regarding the true identity of the training samples.

3. Noise introduced by instrumentation.

4. Actual overlap of pattern classes in the feature space, suggesting use of a "maximum likelihood" or "minimum risk" decision procedure.

Many statistical pattern recognition techniques make use of the conditional probability density functions associated with the pattern classes, and such an approach is assumed in Chapters 2 through 6 of this report. The densities are usually unknown a priori and must be estimated from a set of observations or patterns of known classification, called training patterns. A useful density estimation method must produce a density estimate which is accurate over the portion of the pattern space which may contain patterns to be recognized and readily implemented in terms of a practical physical pattern recognition system. Adequately meeting both of these requirements for any but the simplest density functions is no trivial matter, as is amply illustrated in the pattern recognition literature.

The terms "parametric" and "nonparametric" are frequently used in the statistical pattern recognition literature to describe certain types of approaches, but the distinction is sometimes not very well defined. In this report, a method is called parametric if it assumes a priori that the true or underlying probability density functions to be estimated are of a particular form. A method is called nonparametric if it does not presuppose the form of the underlying densities--but this does not preclude prior specification of

the form of the estimate. For example, a familiar parametric approach is to assume that the underlying densities are Gaussian and to use the training patterns to estimate the means and covariances [1]. A multidimensional histogram of the training data constitutes a conceptually simple nonparametric density estimate. Some more sophisticated nonparametric methods are reviewed in Chapter 2.

Parametric approaches often have the advantage of providing compact closed form analytical expressions for the density estimates. These expressions can then be evaluated easily to accomplish the recognition of a pattern of unknown classification. "Real world" problems rarely fit into such a simplified framework, however; there is rarely sufficient a priori information to permit accurate prespecification of the form of the underlying densities. In such cases, nonparametric methods which make less restrictive assumptions about the densities seem more appropriate and may result in more accurate estimates. It will be seen in subsequent discussions, however, that care must be exercised to ensure that the resulting densities can be utilized in a practical manner.

The nonparametric density estimation approach investigated in Chapters 3 through 6 aims at general applicability and realistic implementation requirements. The only critical assumption made is that the training samples are available for preprocessing on a digital computer. The data are analyzed by a simple mode discrimination technique and the results used in conjunction with a consistent probability density estimator to synthesize a recognition algorithm. A polynomial approximation of the density estimator permits, where

necessary, the trading of estimation accuracy for implementation simplicity.

1.2 Linguistic Pattern Recognition

As sentence structure is a principal object of the study of natural and artifical languages, so pattern structure is the principal concern of linguistic pattern recognition. This approach to pattern recognition is based on the observation that for many pattern recognition problems of current interest, particularly those in which the data are essentially nonnumerical (eg., pictorial data), structural relationships within the patterns constitute the features of significance. The methodologies of formal linguistics provide a convenient formalism for representing and operating on structural ("syntactic") relationships. Chapter 7 of this report contains an overview and appraisal of the state-of-the-art of linguistic pattern recognition. Chapters 8 and 9 consider some aspects of existing formal language theory and some generalizations thereof which appear of particular value for linguistic pattern recognition. These include the use of automata as relatively simple recognizers for pattern languages, use of the programmed grammar as a simple but powerful formalism for the generation and linguistic analysis of patterns, and the introduction of probabilistic mechanisms into linguistic formalisms in order to better deal with the statistical aspects of real-life pattern recognition problems.

CHAPTER 2

THE NONPARAMETRIC APPROACH: BACKGROUND

2.1 Basic Assumptions, Notation and Terminology

The familiar model for pattern recognition systems is assumed (Figure 2.1) [2], [3]. It is further assumed that a suitable set of pattern features have been selected by appropriate means [4] - [6], and are available as input to the decision mechanism. The probability density functions characterizing the pattern classes are estimated based on these features.

There are assumed to be K pattern classes: ω_i, $i = 1, 2, \ldots, K$. Each pattern is represented by a feature vector $X = (x_1, x_2, \ldots, x_N)$ in N-dimensional Euclidean space. There are available T_i (a finite number of) training patterns associated with pattern class ω_i, the entire training set represented by $\{Y_{ij} \mid i = 1, 2, \ldots, K;$ $j = 1, 2, \ldots, T_i\}$.

Where no confusion will result, the notation may be simplified by dropping the subscript denoting the class. This will be done, for example, where the class is arbitrary.

For concreteness, a Bayesian (minimum expected risk) decision strategy will be assumed [7], with unit cost for misclassification and zero cost for correct classification. The classification rule is then: Decide $X \in \omega_i$ if

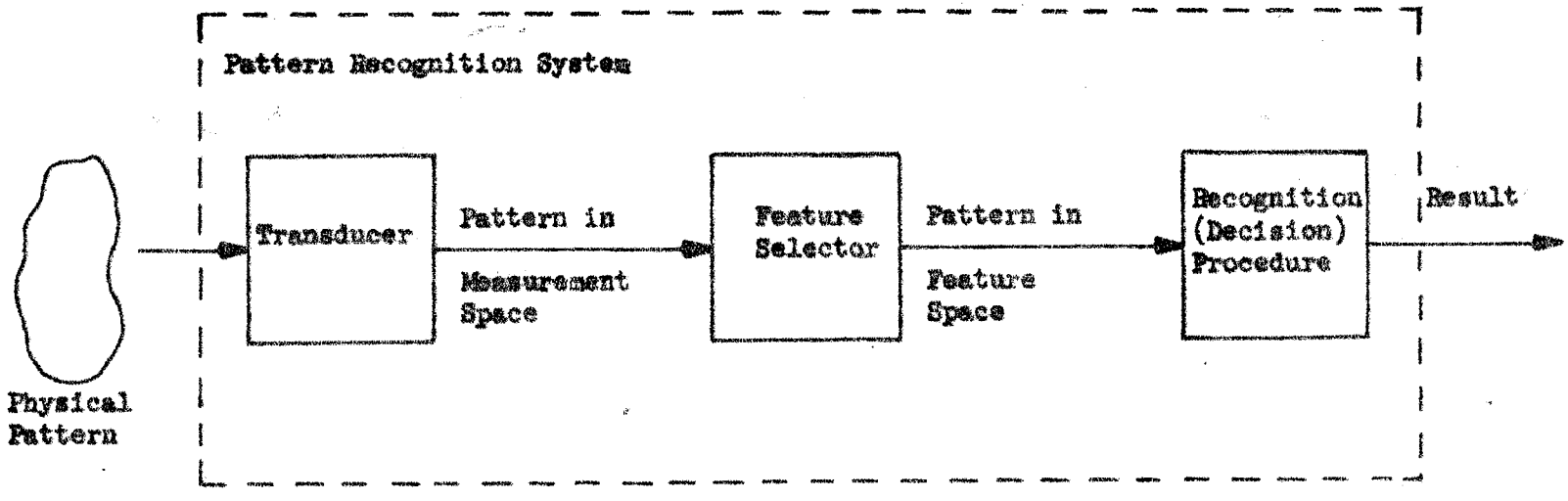


Figure 2.1 Model for a Pattern Recognition System

$$P_i\, p(X \mid \omega_i) \geq P_j\, p(X \mid \omega_j), \quad i, j = 1, 2, \ldots, K \tag{2.1}$$

where P_i is the a priori probability of class ω_i and $p(X \mid \omega_i)$ is the conditional probability density at X given that X is a pattern in ω_i.

The problem of estimating a probability density p(X) (the conditional notation will be dropped for the balance of this chapter) may be stated as follows: Find an estimate $\hat{p}(X)$ of p(X) given a set of T observations, random vectors in N-dimensional space,

$$\left\{Y_i = (y_{i1}, y_{i2}, \ldots, y_{iN})' \mid i = 1, 2, \ldots, T\right\}$$

(training patterns in pattern recognition parlance) which are independent and identically distributed with probability density p(X).

2.2 Literature Review

Methods for nonparametric density estimation which have been described in the literature may be grouped roughly as 1) smoothing methods, 2) stochastic approximation and similar methods, and 3) others.

2.2.1 Smoothing Methods

Smoothing methods estimate the probability density p(X) by estimators of the form

$$\hat{p}(X) = \frac{1}{T} \sum_{i=1}^{T} g(X - Y_i). \tag{2.2}$$

The function g(X) is often referred to in the literature as a weighting function. The term smoothing function also seems quite appropriate in view of the following interesting interpretation of

(2.2) suggested in [8]. The one-dimensional version of (2.2) can be written in the form

$$\hat{p}(x) = \int_{-\infty}^{\infty} g(\xi) \left[\frac{1}{T} \sum_{i=1}^{T} \delta(x - y_i - \xi) \right] d\xi, \tag{2.3}$$

where $\delta(x)$ is the delta function or unit impulse function which concentrates unit mass at $x = 0$, and $y_1, y_2, \ldots, y_T$ are the one-dimensional observations. Now let

$$p_E(x) = \frac{1}{T} \sum_{i=1}^{T} \delta(x - y_i),$$

which, for want of a better term, will be called the <u>empirical estimate</u> of the density $p(x)$.* The empirical estimate is formed by concentrating mass $1/T$ at each of the points where an observation occurs (note the similarity to a conventional histogram). Equation (2.3) can then be written as

$$\hat{p}(x) = \int_{-\infty}^{\infty} g(\xi) p_E(x - \xi)\, d\xi, \tag{2.4}$$

---

*by analogy with the <u>empirical distribution</u> [9], [10]

$$F_E(x) = \frac{1}{T} \sum_{i=1}^{T} u(x - y_i)$$

where $u(x)$ is the unit step function. Note however that although $F_E(x)$ is a pointwise consistent estimator for the distribution function $F(x)$, $p_E(x)$ is not a pointwise consistent estimator for the density function $p(x)$.

which is simply the convolution of $g(x)$ and $p_E(x)$. In other words, the estimate may be interpreted as the response of a low-pass "spatial filter" or smoothing filter with impulse response $g(x)$ when the input is $p_E(x)$. Thus $g(x)$ "smooths" the empirical estimate. A similar development can be made for the multidimensional case by defining a multidimensional spatial filter in the obvious way.

The majority of papers dealing with this form of estimator are limited to consideration of the one-dimensional case. Rosenblatt [11], Whittle [12], Parzen [13], and Watson and Leadbetter [14] have investigated the problem of selecting $g(x)$ so as to minimize various error criteria (principally mean integral square error). Unfortunately the solutions are highly dependent on a priori knowledge of the form of the density to be estimated, information here assumed unavailable (in [14] it is shown that the optimal $g(x)$ may be obtained by inverting an expression involving the characteristic function of $p(x)$). Furthermore, the smoothing functions obtained are generally so complicated that even if sufficient a priori information could be assumed, the resulting estimators would be of questionable practical value.

With these problems in mind, some authors have undertaken to find suboptimal density estimates using restricted forms of smoothing functions. For example, Cooper and Tabaczynski [9] have assumed $g(x)$ to be linear in x. Even in this case, however, the best estimate cannot be achieved without complete a priori knowledge of the density to be estimated.

Rather more general and more useful results have been established by Parzen [13] for the one-dimensional case and generalized by Murthy [15] to multidimensional estimators. In particular, Murthy has proven the following theorem which is important with respect to the density estimator to be developed in this report.

Theorem 2.1 [15]: Assume that

1. the observations $\{Y_i\}$ have a common multivariate distribution $F(X)$ representable as

$$F(X) = F_1(X) + F_2(X)$$

where $F_1(X)$ is an absolutely continuous N-variate distribution and $F_2(X)$ is a distribution with its whole mass concentrated in a set of isolated discrete mass points;

2. the function $g(X)$ satisfies the following

$$g(X) \geq 0$$

$$g(x_1, x_2, \ldots, x_N) = g(\pm x_1, \pm x_2, \ldots, \pm x_N)$$

$$g(x_1, x_2, \ldots, x_N) \leq g(y_1, y_2, \ldots, y_N)$$

if $x_i \geq y_i$, $i = 1, 2, \ldots, N$;

3. $\{r_i\} \triangleq \{r_i(T)\}$, $i = 1, 2, \ldots, N$ are N sequences of non-negative constants depending on T in such a way that

$$\lim_{T\to\infty} r_i(T) = 0 \qquad i = 1, \ldots, N \tag{2.5}$$

and

$$\lim_{T\to\infty} T \prod_{i=1}^{N} r_i(T) = \infty .$$

For convenience of notation, define

$$X/R \triangleq \left[\frac{x_1}{r_1}, \frac{x_2}{r_2}, \ldots, \frac{x_N}{r_N} \right].$$

Then

$$\hat{p}(X) = \left[T \prod_{i=1}^{N} r_i(T) \right]^{-1} \sum_{i=1}^{T} g\left[(X - Y_i)/R(T) \right]$$

is a consistent estimate of $p(X)$ at every point of continuity of $p(X)$ and $F(X)$.

This theorem requires only some fairly rudimentary a priori knowledge regarding the density to be estimated. However, it deals only with asymptotic properties of a general class of estimators. The density estimation method to be investigated herein involves one estimator in this class. A major effort will be directed toward specifying the smoothing parameters, $r_i(T)$, $i = 1, 2, \ldots, N$ in such a way as to satisfy Theorem 2.1 in the process of obtaining an accurate density estimate for finite T.

One effort to utilize Murthy's result has been carried out by Specht [8], who employed the smoothing function

$$g(X) = \frac{1}{(2\pi)^{N/2} r^N} \exp\left(-\frac{1}{2r^2} X'X \right). \tag{2.7}$$

Specht has assumed

$$r_1(T) = r_2(T) = \ldots = r_N(T) = r \tag{2.8}$$

and used trial-and-error to optimize the single smoothing parameter. The estimator proposed in Chapter 3 is essentially a generalization of Specht's estimator for which the $r_i(T)$ are not assumed equal and must be systematically determined.

2.2.2 Stochastic Approximation and Similar Methods

This group of methods is intended to encompass a wide range of methods which utilize estimators of the form

$$\hat{p}(X) = \sum_{i=1}^{m} c_i \phi_i(X) \tag{2.9}$$

where m may be finite or infinite (in practice, of course, m must be finite). Included in particular are stochastic approximation methods [16] and potential function methods [17], [18], which, although differing somewhat in their philosophy of approach, are theoretically very closely related [19], [20]. These and other methods included in this category [21], [22] differ principally in their specifications for the set of "ϕ-functions" and the details for calculating the coefficients c_i. Unfortunately the one statement perhaps most applicable to all of these methods in general is that, although some of them have theoretically appealing characteristics, they are quite difficult to apply in practice because of problems associated with finding suitable ϕ-functions, particularly for multidimensional applications ([16] contains some interesting empirical studies along these lines).

2.2.3 Other Methods

There are a number of additional nonparametric density estimation methods which deserve mention but cannot be grouped as readily as those discussed above.

An interesting and conceptually simple approach which yields a multivariate density estimate is discussed in [23]. The local

density at a point X is computed by a process which amounts roughly to counting the k observations nearest to X and dividing by the smallest hyperspherical volume centered on X and containing the k neighbors. If k is a function of the total number of observations for the density and satisfies some relatively simple restrictions (much like those for the smoothing parameters in Theorem 2.1), the estimate can be shown to be consistent. Again, however, only asymptotic properties of the estimator have been established and it is not clear that it can be expected to perform well in general for a limited number of training samples.

Patrick and Bechtel [24] have proposed a histogram technique for density estimation in which the bin size is adaptively determined so as to minimize the memory required to store the histogram. The method is directly applicable only in the one-dimensional case, but the authors have described methods for mapping multidimensional space onto the real line so that their method can be applied to multi-dimensional data.

Finally, several nonparametric pattern recognition methods, while not explicitly estimating the density functions associated with the pattern classes, depend implicitly (either theoretically or at least philosophically) on such an estimate. Stochastic approximation methods which estimate the decision boundaries directly [16] are good examples of this. Another is a method due to Fu and Henrichon [25] in which the decision boundaries are defined by the surfaces at which $p(X \mid \omega_i) - p(X \mid \omega_j)$ changes sign, for any pair of classes ω_i and ω_j which are adjacent in the feature space.

CHAPTER 3

A VERSATILE NONPARAMETRIC DENSITY ESTIMATOR

Existing nonparametric density estimation techniques such as those reviewed in the preceding chapter are often difficult to apply in practice because of one or more of the following factors.

1. The estimators cannot be expected to provide accurate density estimates based on relatively limited sets of training patterns.

2. Extension of one-dimensional methods to multiple dimensions results in impractical memory and/or computational requirements.

3. Accurate density estimation requires specification of certain parameters which depend on unavailable a priori information about the density.

4. The methods are in some respect iterative but stopping criteria are not well defined.

The nonparametric method to be proposed here belongs to the category of smoothing methods described in Chapter 2. It may be used to estimate complex multidimensional density functions (only a reasonable degree of regularity or smoothness is assumed); all necessary parameters may be determined from the training patterns; and the method is noniterative. For cases in which the memory and computational requirements become excessive, a polynomial approximation of the estimator, described in Chapter 5, may allow the user to find a suitable trade-off between these requirements and the accuracy of the estimate.

3.1 The Estimator

The smoothing function

$$g(X) = \frac{1}{(2\pi)^{N/2} |S|^{1/2}} \exp\left[-\frac{1}{2} X' S^{-1} X\right] \tag{3.1}$$

leads to the estimator

$$\hat{p}(X) = \frac{1}{(2\pi)^{N/2} |S|^{1/2}} \cdot \frac{1}{T} \sum_{i=1}^{T} \exp\left[-\frac{1}{2} (X-Y_i)' S^{-1} (X-Y_i)\right] \tag{3.2}$$

where N is the dimensionality of the feature space; Y_i, i = 1, 2, ..., T are independent, identically distributed observations or training patterns; and S is an N x N matrix of smoothing parameters. S will be restricted herein to be a diagonal matrix of the form

$$S \triangleq \text{diag}\,[RR'] \triangleq \begin{bmatrix} r_1^2 & & & 0 \\ & r_2^2 & & \\ & & \ddots & \\ 0 & & & r_N^2 \end{bmatrix} \tag{3.3}$$

where R = R(T) is an N-vector of smoothing parameters as defined in Theorem 2.1. It is readily verified that (3.1) and (3.3) define a smoothing function which satisfies the requirements of Theorem 2.1 provided R(T) is a suitably defined vector function of T. Therefore (3.2) is a consistent estimator of p(X) at every point of continuity of p(X) and its distribution function F(X).

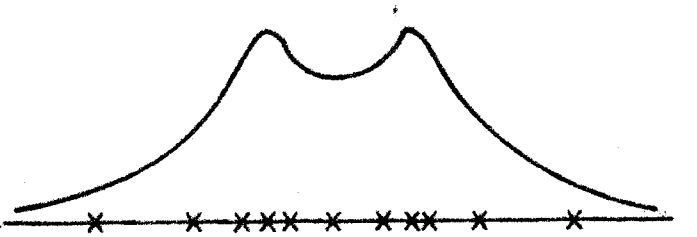
The choice of smoothing function for study was somewhat arbitrary in the same sense that the use of mean square error, say, is an arbitrary but convenient choice of error criterion; i.e., it

satisfies the requirements of existing theory and leads to tractable results. At the outset of the investigation the choice was logical because it offered the possibility of generalizing and improving some earlier efforts at practical application which showed promising results [8]. Further justification was provided by the fact that the regularity properties of the estimator (existence and smoothness of derivatives) are similar to those of the density functions associated with the data observed in real situations to which this pattern recognition technique could be applied. There are other smoothing functions with similar characteristics, however, which could be considered in further studies of this approach. A short table of such functions may be found in [13].

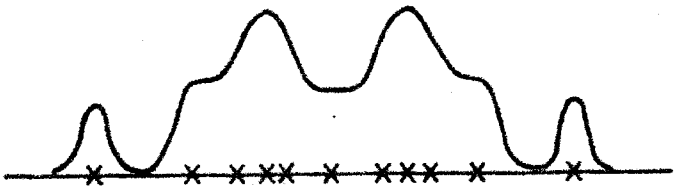
A one-dimensional illustration will give some insight into how the estimate is effected by the smoothing parameters. The estimator for the one-dimensional case is

$$\hat{p}(x) = \frac{1}{\sqrt{2\pi}\, r} \cdot \frac{1}{T} \sum_{i=1}^{T} \exp\left[-\frac{1}{2}\left(\frac{x-y_i}{r}\right)^2\right]. \tag{3.4}$$

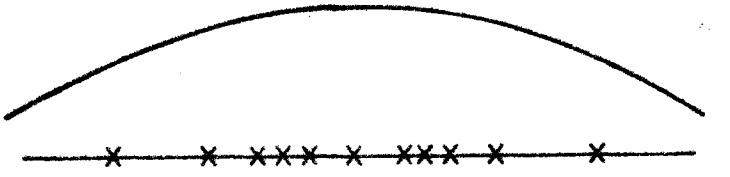
The smoothing parameter r controls the rate of exponential decay of influence of each pattern with increasing distance in the feature space from that pattern. For small values of r the estimate may vary between training patterns more rapidly than for larger values. The situation is illustrated in Figure 3.1, in which it is assumed that the set of observations (denoted by X's along the horizontal axis) is drawn from a population with hypothetical density p(x) as shown in (a). If r is chosen too small, the density will be undersmoothed as sketched in (b), which amounts to overconfidence that the



(a) Hypothetical Density p(x)



(b) Undersmoothed Estimate $\hat{p}(x)$



(c) Oversmoothed Estimate $\hat{p}(x)$

Figure 3.1 Smoothing Parameter Effects

set of observations adequately represents the density. If r is chosen too large, detailed information present in the observation set will be oversmoothed as in (c), which amounts to undue lack of confidence in the observations. If r is properly chosen somewhere between these extremes, a reasonable estimate minimizing some measure of the estimation error may be achieved.

Theorem 2.1 guarantees that for any selection of smoothing parameters satisfying (2.5) and (2.6), the estimate may be expected to become identical with the true density in the limit as the number of observations T becomes infinite. But concern here is with the practical situation in which only a finite (even small) number of observations is available. Specht [8] studied the case in which a single smoothing parameter is used in (3.2) for all pattern classes and all features; i.e., $r_1 = r_2 = \ldots = r_N$ for all classes. In such a simple case he was able to use trial-and-error to determine the optimal parameter value. The experimental work presented in Chapter 6 illustrates that allowing different parameters for different classes and features (indeed even for different subclasses with a given class) can significantly improve both the estimation accuracy and pattern recognition performance.* But then trial-and-error optimization is no longer feasible and alternative means must be found for determining the smoothing parameters. One would hope to

*Parzen [13] showed that for a particular error criterion (pointwise mean square error) and $T \to \infty$, the optimal smoothing parameter is a function of X. But this generally leads to impractical implementation requirements.

be able to determine the smoothing parameters from the training patterns, a possibility considered in the next section.

3.2 Smoothing Parameter Determination

For the estimator defined by (3.2) and (3.3), N smoothing parameters must be specified for each density to be estimated, a total of K·N parameters for a problem involving K pattern classes.

It should be clear from the one-dimensional illustration that the optimal smoothing parameters are apparently related to the dispersion of the patterns in the feature space. More smoothing (larger values) is required when the patterns are spread thinly than when they are tightly clustered. A familiar measure of dispersion is variance. The unbiased sample variance of each feature is given by

$$s_i^2 = \frac{1}{T-1} \sum_{j=1}^{T} (y_{ji} - \bar{y}_i)^2$$

$$= \frac{1}{T-1} \sum_{j=1}^{T} (y_{ji} - \frac{1}{T} \sum_{k=1}^{T} y_{ki})^2, \quad i = 1, 2, \ldots, N \qquad (3.5)$$

The square roots of the feature sample variances will be referred to as the feature sample standard deviations. For large T, s_i^2 is approximately equal to the average squared distance, measured along the i^{th} feature axis, of the observations from the sample mean. Looking at it another way, some manipulation shows that the sample variance can also be written in the form

$$s_i^2 = \frac{1}{T(T-1)} \sum_{k=2}^{T} \sum_{j=1}^{k-1} (y_{ji} - y_{ki})^2 = \frac{1}{2} \overline{(d_i^2)} \qquad (3.6)$$

where $\overline{(d_i^2)}$ is the average of the squared distances, measured along the i^{th} feature axis, between all pairs of observations. This relation provides an intuitive feeling for how variance expresses pattern dispersion.

However, recalling the consistency requirements it is immediately apparent that the feature sample variances (or standard deviations) cannot be the smoothing measures sought. As the number of observations (size of the training set) increases, the feature sample variances approach constant values equal to the true variances associated with the population from which the observations are drawn (assuming a finite variance population, of course). This violates (2.5) which specifies that the smoothing parameters must vanish as $T \to \infty$.

A smoothing measure which does satisfy the consistency requirements while utilizing the feature sample standard deviations will be proposed. First it will be convenient to define an effective sample volume for a set of observations. A function of this volume and of the number of observations will then be introduced which amounts roughly to a measure of pattern density over the effective sample volume.

The volume of the feature space over which the probability density in question is nonzero could, at least in theory, be infinite. However, it is generally the case that a very large proportion of the observations are likely to fall near the mean of the population. In the one-dimensional case, this property is expressed mathematically by the Bienaymé-Tchebycheff inequality [26],

$$P\left[\,|x - \bar{x}| \geq k\sigma\right] \leq \frac{1}{k^2}$$

where σ is the standard deviation from the mean. Accordingly, a finite effective sample volume will be defined, based on the ellipsoid of concentration, which depends only on the feature standard deviations and the dimensionality of the feature space.

The concept of ellipsoid of concentration arises as follows [26]: Consider a one-dimensional random variable x with mean $\bar{x}$ and standard deviation σ. If z is another random variable uniformly distributed over the interval

$$I = (\bar{x} - \sigma\sqrt{3},\ \bar{x} + \sigma\sqrt{3}),$$

then z has the same mean and standard deviation as x. Therefore, the interval I provides a geometrical characterization of the concentration (or dispersion, which is just the inverse relation to concentration) of the distribution of x about its mean. The length of the interval, $L_I = 2\sigma\sqrt{3}$, is a scalar which numerically characterizes the the concentration or dispersion.

To generalize to N dimensions, let the random vector X have zero mean and covariance matrix $\Lambda = [\lambda_{ij}]$. If Z is another random vector uniformly distributed over the volume of the ellipsoid

$$Q(Z) = \sum_{i=1}^{N} \sum_{j=1}^{N} \frac{|\Lambda_{ij}|}{|\Lambda|} z_i z_j = N + 2 \qquad (3.7)$$

where $|\Lambda_{ij}|$ is the cofactor of λ_{ij}, then Z also has zero mean and covariance matrix Λ. This ellipsoid, the ellipsoid of concentration of any distribution with zero mean and covariance Λ, serves as a geometrical characterization of the concentration (or dispersion) of the multivariate distribution about the mean (zero mean as defined

above, but the generalization to arbitrary mean is not needed here since the volume of the ellipsoid is independent of the mean). The volume of the ellipsoid of concentration, which provides a numerical characterization of the dispersion of the multivariate distribution, is given by

$$V_Q = \frac{(N+2)^{N/2}\,\pi^{N/2}}{\Gamma(N/2+1)}\,|\Lambda|^{1/2}$$

$$= \frac{(N+2)^{N/2}\,\pi^{N/2}}{\Gamma(N/2+1)}\cdot\sigma_1\,\sigma_2\cdots\sigma_N\,|P|^{1/2} \qquad (3.8)$$

where $P = [\rho_{ij}]$ is the correlation matrix, i.e.,

$$\rho_{ij} = \frac{\lambda_{ij}}{\sigma_i\,\sigma_j}\,.$$

This volume will be defined to be the <u>effective volume of the distribution</u> and written hereafter as

$$V_Q = C(N)\,|\Lambda|^{1/2} \qquad (3.9)$$

where

$$C(N) = \left[(N+2)^{N/2}\,\pi^{N/2}\right] / \Gamma(N/2+1).$$

If instead of the actual population covariance matrix Λ, the sample covariance matrix (the unbiased estimate of Λ based on the training samples) is used, the volume will be called the <u>effective sample volume</u>, denoted $V_{\tilde{Q}}$.

Now, it will be hypothesized that the smoothing parameters may be given by an expression of the form

$$r_i = k_1\,c(N)\,s_i\,T^{-1/k_2 N}, \qquad i = 1, 2, \ldots, N \qquad (3.10)$$

where

k_1 and k_2 are constants,

$c(N)$ is a constant which depends only on the dimensionality N,

s_i is the feature sample standard deviation,

T is the number of observations in the training set.

This expression satisfies the consistency requirements of Theorem 2.1, since for $k_1 > 0$ and $k_2 > 1$

$$\lim_{T\to\infty} r_i = k_1 c(N) \lim_{T\to\infty} s_i T^{-1/k_2 N} = 0 \tag{3.11}$$

and

$$\lim_{T\to\infty} T \prod_{i=1}^{N} r_i = (k_1 c(N))^N \lim_{T\to\infty} s_1 s_2 \cdots s_N \, T^{(k_2-1)/k_2} \tag{3.12}$$

$$= \infty .$$

An interesting choice for the dimensionality constant $c(N)$ is

$$c(N) = [C(N)]^{1/N}$$

$$= \frac{[(N+2)\,\pi]^{1/2}}{[\Gamma(N/2+1)]^{1/N}} \tag{3.13}$$

for then

$$\prod_{i=1}^{N} r_i = k_1^N \frac{V_{\tilde{Q}}}{T^{1/k_2}} \tag{3.14}$$

where $\tilde{Q}$ is the effective sample volume, here assuming uncorrelated features. According to this choice, the overall effect of the smoothing parameters is proportional to the effective sample volume--a measure of dispersion--and inversely proportional to a power of the number of samples.

It would be naive to expect that a universally optimal set of constants k_1 and k_2 could be found, because the variation of the optimal smoothing parameters is known to depend on the precise form of the underlying density function [11]. But the results of Chapter 6 indicate that (3.10) and (3.13) do indeed provide a smoothing relation that is useful in some interesting and practical cases, and that suitable values for the parameters k_1 and k_2 can be obtained experimentally.

Some final remarks concerning the proposed form of the smoothing parameters: It has been seen that the consistency conditions restrict the nature of the smoothing parameter dependence on the number of observations; the form chosen is one of the simplest forms satisfying these conditions. It is desired to force the use of sample variances to measure dispersion because variances are easily calculated, and the calculation does not require storage of the entire training set. Since the ellipsoid of concentration concept encompases both feature variance and dimensionality effects, it seems a natural way to account for these effects. Still the validity and, above all, the utility of this "smoothing hypothesis" rest on the experimental results of Chapter 6.

CHAPTER 4

THE PROBLEM OF MULTIPLE MODES

The method described in the preceding chapter for determining the smoothing constants depends on the assumption that variance is a reliable measure of dispersion. But even more is required: Since the objective is to smooth the density estimate between adjacent observations, a local measure of dispersion is needed.

As previously noted, the sample variance is proportional to the average squared distance between pairs of observations. When the observations are clustered in a single region of high pattern density, or mode,* the average squared distance between pairs of patterns is a good measure of dispersion. In multimodal situations, however, this is no longer the case; the distances between patterns associated with different modes have an exaggerating influence on the overall variance. But if the modes can be isolated, then a set of smoothing parameters may be calculated for each mode. The estimate given by (3.2) is then rewritten as:

$$\hat{p}(X) = \frac{1}{(2\pi)^{N/2}} \cdot \frac{1}{T} \sum_{i=1}^{T} |S_i|^{-1/2} \exp\left[\frac{1}{2}(X-Y_i)' S_i^{-1}(X-Y_i)\right] \tag{4.1}$$

*This usage of the term "mode" varies from the conventional statistical definition (any maximum point of a density function) but implies nearly the same thing. "Cluster" is a synonymous term.

The smoothing parameter matrix associated with a given observation (as determined by the subset or mode to which that observation belongs) is used in determining the contribution of that observation to the density estimate.

The problems involved in making this approach useful are fairly obvious. First, each training set must be partitioned by a suitable algorithm into subsets corresponding to distinct modes. Second, since it is clearly desirable, from a computation and memory requirement point of view, to minimize the number of subsets within each training set (and hence minimize the number of smoothing parameter matrices which must be stored), the partitioning algorithm must include a rule for determining when a sufficient degree of partitioning has been achieved. These are the problems to be dealt with in this chapter.

4.1 Methods for Mode Discrimination

The term mode discrimination will be used here to refer to any of a broad class of methods for partitioning data sets into distinct subsets under some measure of similarity. A wide variety of such methods has been studied (see [37] for an extensive bibliography), but most mode discrimination methods may be described in terms of the following steps.

1. Partition the data set using an appropriate similarity criterion.

2. Test the partition to determine whether it is significant, i.e., whether the subsets are sufficiently distinct. If so, let the partition stand; if not, merge any subsets which are not sufficiently distinct.

3. Repeatedly partition the subsets created, testing at each step as in step 2, until no further permanent subdivisions result, or until some other stopping criterion is satisfied.

A specific mode discrimination algorithm results from this general procedure when a similarity measure, distinctness test, and stopping rule are supplied.

The following mode discrimination method, an adaptation of a method described by Mattson and Dammann [28] illustrates some significant points. It is a form of principal component analysis [29]. Let $X_k = [x_{k1}, x_{k2}, \ldots, x_{kN}]$, $k = 1, 2, \ldots, T$ be a set of data to be analyzed for multimodality. Define the scalar similarity measure s_k, evaluated for the k^{th} observation, by

$$s_k = W' X_k = \sum_{i=1}^{N} w_i x_{ki}$$

where W is a vector of weighting factors as yet unspecified. For a given vector W, the value of s_k may be plotted along the real line (s axis). It is desired to determine W such that if the data set is bimodal the bimodality will be clearly exhibited by the s axis plot of all members of the data set. In other words, patterns in distinct modes will yield distinctly different values for s_k. Thresholding can then be used to separate the observations into their respective modes.

The desired condition on W is achieved by maximizing the function

$$\lambda = \left[\sum_{k=1}^{T} (s_k - \bar{s})^2 \Big/ \sum_{i=1}^{N} w_i^2 \right] = W' AW / W' IW \qquad (4.2)$$

where

$$\bar{s} = \frac{1}{T} \sum_{k=1}^{T} s_k,$$

$$A = \left[a_{ij}\right] = \left[\sum_{k=1}^{T} (x_{ki} - \bar{x}_i)\,(x_{kj} - \bar{x}_j) \right]$$

(Note that the A matrix is proportional to the sample covariance matrix.) The function in (4.2) is maximized if W is chosen proportional to the eigenvector corresponding to the largest eigenvalue of A. If it is possible to display the data set as two modes on the s axis, this selection of W will do so.

The distinctness test associated with this approach might require, for instance, that a prescribed length of the s axis between modes contain no sample points (or fewer than some specified number of sample points). The stopping rule might simply be that no further partitions are possible which yield significantly distinct modes.

This method has the advantage that the data set members may be processed serially, thus avoiding the need to store them in direct access memory. On the other hand, it has some significant disadvantages. Calculation of the largest eigenvalue and its corresponding eigenvector are relatively complex computations. In addition, it is not difficult to imagine data configurations involving, say, three modes for which this scheme would not be able to detect any multiplicity of modes. The method also tends to perform poorly whenever the patterns are rather loosely clustered within the modes; i.e., when the intramode dispersion is significant compared to the intermode dispersion.

An interesting adaptation of this approach is described in Fu and Henrichon [25]. In order to get a comprehensive modal analysis, the data is sequentially mapped onto each eigenvector. In each case, a one-dimensional technique, based essentially on the Loftsgaarden and Quesenberry density estimator [23], is applied to estimate the extrema of the density and hence isolate the modes. Some user-selected thresholds are required, but if these are chosen properly the method appears to be quite effective. Of course, the computations required by this method are at least as complex as those described above. In addition, several passes through the data are required.

Other disadvantages common to many mode discrimination methods include: Restrictions on the probability distributions involved, impractical computation time, sensitivity to the order in which the patterns are observed, sensitivity to noise in the data, and, for iterative methods, slow convergence to the final result. The method proposed in the next section and used to obtain the experimental results presented in Chapter 6 alleviates or circumvents entirely all of these problems, improvements which may be attributed in large measure to the nonstringent requirements of the task at hand: It is unnecessary to detect and isolate every subset of the data which is in some measure (however slight) dissimilar from all other subsets; it is sufficient to structure the data just enough so that gross exaggerations of feature variance are avoided. Whenever such exaggerations do not result, subsets should be lumped together in order to minimize the number of smoothing parameter matrices which must be calculated and stored.

4.2 A Mode Discrimination Algorithm

The algorithm to be used here for mode discrimination consists of the following steps (refer to Figure 4.1):*

1. Initialization: Select two distinct but otherwise arbitrary observations from the data set under consideration. These observations serve as initial mode centers.

2. Mode assignment: Calculate the Euclidean distance of each observation from each mode center, and assign each observation to the mode with the nearest mode center (note that Euclidean distance is the similarity measure to be used).

3. Mode migration: Calculate the mean (center of gravity) of the patterns assigned to each mode in step 2: (a) If these means are identical with the mode centers used in step 2, go to step 4. (b) Otherwise, replace the old mode centers by the new means and return to step 2.

4. Merging: A new set of modes has now been tentatively established. Test each pair of modes using the distinctness test given later in this section. Merge any modes which are not significantly distinct under this test. If t_i and t_j are the numbers of observations in two modes and M_i and M_j are the respective mode centers, then the merged mode center is given by

$$M = \frac{t_i M_i + t_j M_j}{t_i + t_j} .$$

*This algorithm was inspired by a method described in [30].

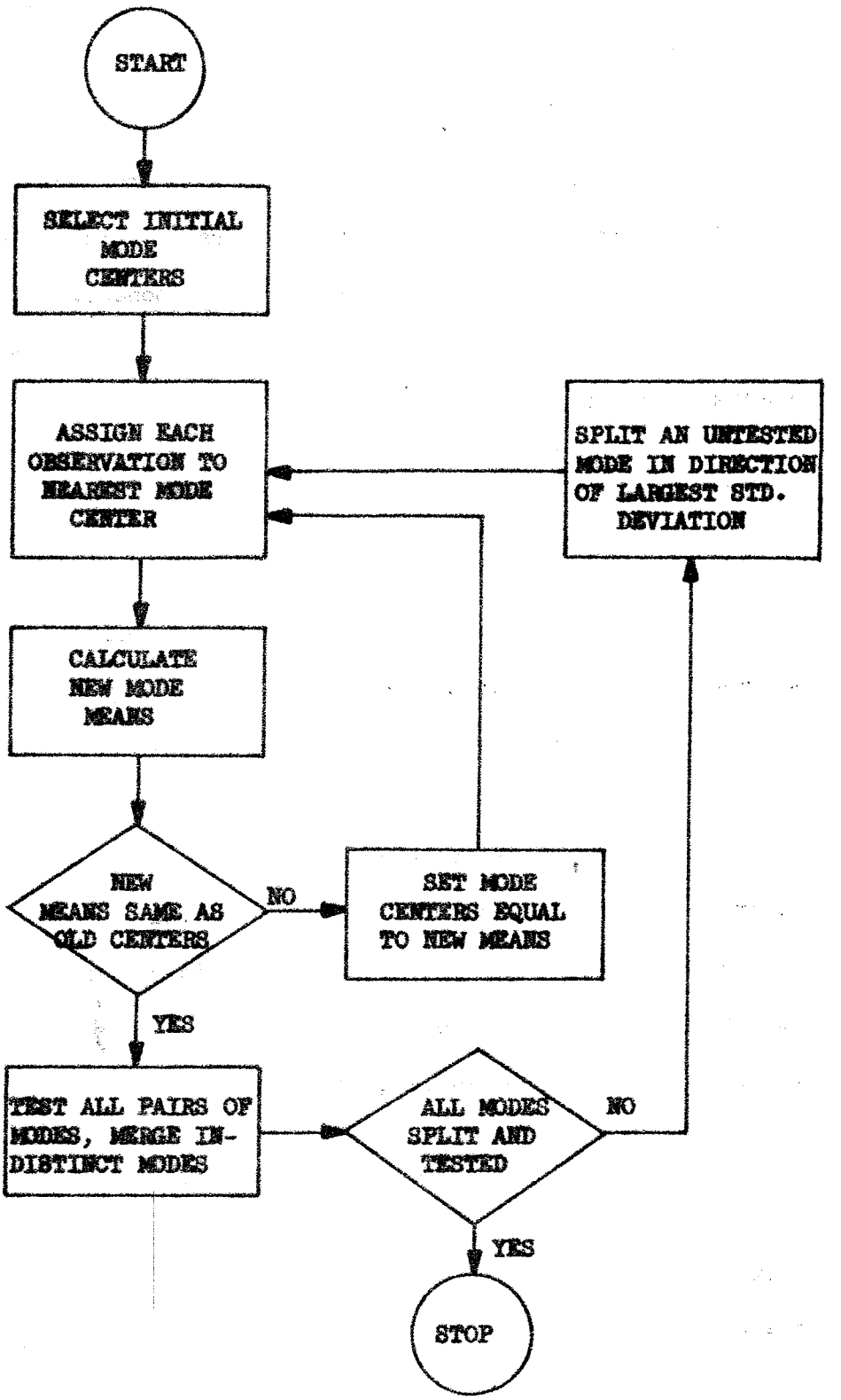


Figure 4.1 Mode Discrimination Algorithm

5. Mode splitting: (a) If all existing modes have not yet been tentatively split and tested, select a mode which has not been so processed and split it by adding and subtracting a small amount from the component of the mode center having the largest standard deviation. Two new tentative mode centers are thus formed which are identical with the old one except in one component. Return to step 2. (b) If all presently existing modes have been split and tested, the analysis is complete.

The similarity measure and stopping rule have been specified in steps 2 and 5 respectively. There remains to be specified the rule for determining in step 4 when two modes are distinct enough to remain separate. Since the smoothing parameters depend on the ellipsoid of concentration, it is satisfying that the ellipsoid of concentration can also provide a useful distinctness test, which will now be developed.

It will be necessary to know the distance from the center of an ellipsoid of concentration to the boundary (surface) of the ellipsoid in a given direction. Since the distance is origin-independent, it may be assumed that the center of the ellipsoid is at the origin of the space. The ellipsoid is given by

$$\sum_{i=1}^{N} \sum_{j=1}^{N} \frac{|\Lambda_{ij}|}{|\Lambda|} x_i x_j = N + 2 . \tag{4.3}$$

A line from the center of the ellipsoid and passing through the point $A = (a_1, a_2, \ldots, a_N)$ is defined by the equation

$$\frac{x_1}{a_1} = \frac{x_2}{a_2} = \cdots = \frac{x_N}{a_N} . \tag{4.4}$$

Let D(A) be the distance from the center of the ellipsoid of concentration to the surface of the ellipsoid along the line through its center and the point A. By solving (4.3) and (4.4) simultaneously, it may be verified that

$$D(A) = \alpha\,(N + 2)^{1/2} \tag{4.5}$$

where

$$\alpha^2 = \frac{\sum_{i=1}^{N} a_i^2}{\sum_{i=1}^{N}\sum_{j=1}^{N} |\Lambda_{ij}|\, a_i a_j}\, |\Lambda| .$$

Now, given two modes with mode centers $M_1 = (m_{11}, m_{12}, \cdots, m_{1N})$ and $M_2 = (m_{21}, m_{22}, \cdots, m_{2N})$, let M be the (undirected) line connecting the mode centers. Denote by D_1 the distance along M from M_1 to the surface of the corresponding ellipsoid of concentration; D_2 is the corresponding distance for M_2 and its ellipsoid. Utilizing the result given above, it is found that

$$D_k = \beta_k (N + 2)^{1/2}, \qquad k = 1, 2 \tag{4.6}$$

where

$$\beta_k^2 = \frac{\sum_{i=1}^{N} (m_{2i} - m_{1i})^2}{\sum_{i=1}^{N}\sum_{j=1}^{N} |\Lambda_{ij}^{(k)}|\, (m_{1i} - m_{2i})\,(m_{1j} - m_{2j})}\, |\Lambda^{(k)}|$$

The superscript (k) on a matrix denotes which mode the matrix is associated with.

Let D_{12} be the distance between mode centers, i.e.,

$$D_{12} = |M_2 - M_1| = \left[\sum_{i=1}^{N} (m_{2i} - m_{1i})^2 \right]^{1/2}. \tag{4.6}$$

The following rule is then formulated based on the distances D_1, D_2, and D_{12}, where D_1 and D_2 are computed by substituting the sample covariance matrix for the population covariance matrix in (4.6).

Rule (distinctness): Modes 1 and 2 as given above will be considered significantly distinct provided

$$\gamma = \frac{D_{12}}{D_1 + D_2} > \gamma_t \tag{4.7}$$

where γ_t is a suitably chosen threshold. (See Figure 4.2.)

Finding a suitable value for γ_t is discussed in Chapter 6.

If the ellipsoids of concentration are to be defined by feature sample variances only (for example, to reduce the computational load), (4.6) reduces to

$$D_k = (N + 2)^{1/2} \left[\sum_{i=1}^{N} (m_{2i} - m_{1i})^2 \Big/ \sum_{i=1}^{N} \frac{(m_{2i} - m_{1i})^2}{\sigma_{ki}^2} \right]^{1/2}$$

where σ_{ki}^2 is the variance of the i^{th} feature for the k^{th} mode.

Any available prior knowledge regarding the probable number of significantly distinct modes may be utilized to reduce the amount of processing required. Suppose there is reason to expect approximately k significantly distinct modes. Then k distinct observations may be selected and used as initial mode centers in step 1 of the algorithm. If k is taken too large, some of the resulting modes will be merged in step 4.

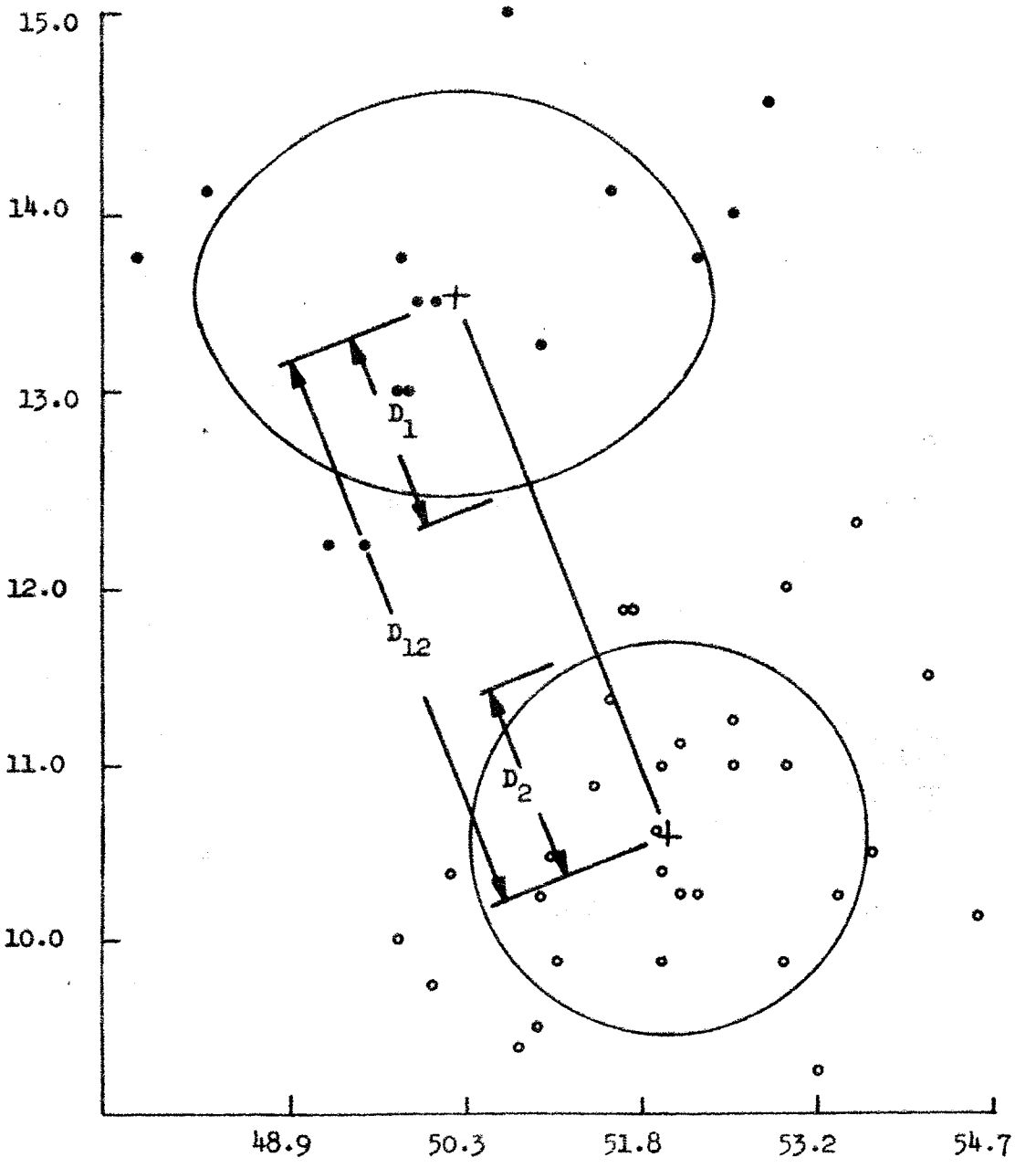


Figure 4.2 Distinct Modes ($\gamma_t = 1$)

Since this algorithm considers the patterns in a sequential manner, the patterns need not be stored in the computer memory. The computations are simple (Euclidean distance rather than, say, eigenvalues and eigenvectors) and convergence is found to be fast enough so that long computation times are avoided. Since all patterns are accounted for on each iteration before adjustment of the mode centers, the algorithm is insensitive to the order in which the patterns are considered. Furthermore, "noisy" patterns (data measurement errors or recording errors) tend to become isolated by the mode splitting procedure. This is easily detected (as distinct modes consisting of very few patterns) and permits the user to take appropriate account of such errors. Since they generally become well isolated from the remainder of the data, such gross errors have no effect on the final partitioning of the "good" data. By contrast, many other techniques are quite sensitive to this sort of error.

The applicability of the proposed mode discrimination technique is well illustrated in Chapter 6.

CHAPTER 5

A POLYNOMIAL APPROXIMATION

A typical pattern classification system utilizing the techniques developed in the preceding chapters would usually be realized by accomplishing the "training" (mode discrimination and smoothing parameter computations) off-line as a design phase and building the results into the system (which might be hardware or simply a computer program). For problems involving feature spaces of relatively low dimensionality and small training sets, the method may be used directly as described. In such cases, the smoothing parameters and training patterns must be stored in the classification system for use during the classification process. But this direct implementation becomes impractical as the product of feature dimensionality times the number of training patterns grows large, since the amount of memory required and the amount of computation per pattern to be classified increases linearly with each of these factors. This chapter presents a method by which the estimator of Chapter 3 (hereafter referred to as "the exponential estimator") may be approximated by a polynomial with coefficients calculated serially (in one pass) from the training patterns, thereby removing the need to store the training set for use in classification and potentially reducing the computations required for each classification.

5.1 Polynomial Expansion of the Exponential Estimator

In this section it is shown how an expression of the form

$$\hat{p}(X) = \frac{1}{(2\pi)^{N/2}T} \sum_{i=1}^{T} |S_i|^{-1/2} \exp\left[-\frac{1}{2}(X - Y_i)'S^{-1}(X - Y_i)\right]$$

may be written as a polynomial series in X having the form

$$\hat{p}(X) = C_0 + C_1x_1 + C_2x_2 + \ldots + C_Nx_N + C_{11}x_1^2 + \ldots + C_{r_1r_2}x_{r_1}x_{r_2} + \ldots + C_{NN}x_N^2 + \ldots + C_{r_1r_2\ldots r_n}x_{r_1}x_{r_2}\ldots x_{r_n} + \ldots$$

Sections 5.2 and 5.3 deal with the practical problems associated with computing the polynomial coefficients and truncating the polynomial series after a finite number of terms.

To make the results developed here more useful in Chapter 6, a constant multiplier δ is factored out of each smoothing parameter so that the density estimator may be assumed to have the form

$$\hat{p}(X) = \frac{1}{(2\pi\delta^2)^{N/2}T} \sum_{i=1}^{T} |S_i|^{-1/2} \exp\left[-\frac{1}{2\delta^2}(X-Y_i)'S_i^{-1}(X-Y_i)\right] \tag{5.1}$$

To simplify the notation, let

$$k_i = (2\pi\delta^2)^{-N/2}|S_i|^{-1/2}.$$

Noting that S_i^{-1} is symmetric, (5.1) can be written as

$$\hat{p}(X) = \frac{1}{T} \sum_{i=1}^{T} k_i \exp\left[-\frac{1}{2\delta^2}(X'S_i^{-1}X - 2Y_i'S_i^{-1}X + Y_i'S_i^{-1}Y_i)\right]$$

$$= \frac{1}{T} \sum_{i=1}^{T} k_i \exp\left[-\frac{1}{2\delta^2} X'S_i^{-1}X\right]\exp\left[\frac{1}{\delta^2} Y_i'S_i^{-1}X\right]$$

$$\cdot\exp\left[-\frac{1}{2\delta^2} Y_i'S_i^{-1}Y_i\right].$$

Now let

$$a_i = -\frac{1}{2\delta^2} X'S_i^{-1} X \qquad \text{(quadratic in X)}$$

$$b_i = \frac{1}{\delta^2} Y_i'S_i^{-1} X \qquad \text{(linear in X)}$$

$$c_i = -\frac{1}{2\delta^2} Y_i'S_i^{-1} Y_i \qquad \text{(independent of X).}$$

Then

$$\hat{p}(X) = \frac{1}{T} \sum_{i=1}^{T} k_i e^{c_i} e^{a_i} e^{b_i} \tag{5.2}$$

The X-dependent exponentials may be written as infinite series. For example:

$$e^{a_i} = 1 + a_i + \frac{a_i^2}{2!} + \frac{a_i^3}{3!} + \ldots + \frac{a_i^n}{n!} + \ldots \tag{5.3}$$

The result of multiplying these series together and collecting terms is as follows.

degree of term	term
0	1
1	b_i
2	$a_i + b_i^2/2!$

degree of term	term
3	$a_i b_i + b_i^3/3!$
4	$a_i^2/2! + a_i b_i^2/2! + b_i^4/4!$
5	$a_i^2 b_i/2! + a_i b_i^3/3! + b_i^5/5!$
.	.
.	.
.	.

This may be summarized by noting that each term of even degree ($n = 0,2,4,6,\ldots$) may be written as

$$\sum_{m=0}^{n/2} \frac{a_i^m b_i^{n-2m}}{m!(n-2m)!},$$

while each term of odd degree ($n = 1,3,5,7,\ldots$) may be written as

$$\sum_{m=0}^{(n-1)/2} \frac{a_i^m b_i^{n-2m}}{m!(n-2m)!}$$

These expressions may be combined to give, in terms of X,

$$\sum_{m=0}^{n^*} \frac{1}{m!(n-2m)!} (X'M_iX)^{n^*-m}(X'\Sigma_iX)^m(X'P_i)^{n^{**}} \qquad (5.4)$$

where

$$n^* = \begin{cases} n/2 & \text{for } n \text{ even} \\ (n-1)/2 & \text{for } n \text{ odd} \end{cases}$$

$$n^{**} = \begin{cases} 0 & \text{for } n \text{ even} \\ 1 & \text{for } n \text{ odd} \end{cases}$$

$$\Sigma_i = -\frac{1}{2\delta^2} S_i^{-1}$$

$$P_i = \frac{1}{\delta^2} S_i^{-1} Y_i$$

$$M_i = P_i P_i' = \frac{1}{\delta^4} S_i^{-1} Y_i Y_i' S_i^{-1}$$

Thus, in the notation defined above, the desired polynomial expansion is:

$$\hat{p}(X) = \frac{1}{T} \sum_{i=1}^{T} k_i e^{c_i} \left[\sum_{n=0}^{\infty} \sum_{m=0}^{n^*} \frac{1}{m!(n-2m)!} (X'M_iX)^{n^*-m} (X'\Sigma_iX)^m \right.$$

$$\left. (X'P_i)^{n^{**}} \right] \qquad (5.5)$$

Admittedly, this is a formidible expression. But it is shown in Section 5.3 that the coefficients of the products and cross-products in $x_1, x_2, \ldots, x_N$ may be computed very systematically.

The form of (5.5) is significant, since it shows that the series coefficients may be calculated by taking the training patterns serially: each c_i, M_i, Σ_i and P_i depends on a single training pattern. This obviates the need to store the entire training set in fast access memory during the coefficient computation procedure.

5.2 Truncation of the Polynomial Expansion

The polynomial expansion of the exponential estimator is, of course, an approximation once a finite number of terms of the series are retained for computing the density estimate, the remainder discarded. To achieve the desired storage and computational

efficiencies sought, the truncation must retain a relatively small number of terms. On the other hand, enough terms must be retained to provide the necessary degree of approximation accuracy. The principal disadvantages associated with use of the polynomial approximation are the difficulties involved in finding a practical trade-off between the number of terms retained and an acceptable level of accuracy of the approximation.

The kinds of difficulties referred to are well documented by Specht [8]. Briefly, when the polynomial is truncated, the quality of the approximation degrades as the distance from the origin of the feature space increases, the rate of degradation varying in an inverse exponential manner with the magnitude of the smoothing parameters. As a result:

1. The origin of the feature space should be shifted to the region in which the sharpest approximation accuracy is required.

2. The number of terms of the polynomial expansion which must be retained to achieve a minimum accuracy over a given portion of the feature space depends on both the extent of that region and the magnitude of the smoothing parameters.

These facts have some important implications. First, since there may be more than one critical region in which accurate approximation is required (this may be the case if there are more than two pattern classes, for instance), it may be necessary to define multiple origins, each with a distinct set of associated polynomial coefficients. A pattern is then classified by using the set of

coefficients corresponding to the nearest origin ("nearest" must be appropriately defined).

Second, although one normally uses as many training samples as possible in order to maximize the accuracy of the density estimate, this advantage may have to be compromised somewhat in using the polynomial approximation (see Section 6.5). This is because as the number of samples increases, the magnitude of the smoothing parameters decreases (see (3.10)), thereby requiring additional polynomial terms to achieve the same quality of approximation. Obviously this can eventually produce the same storage and computational difficulties which the polynomial approximation was intended to alleviate.

Clearly, then, achieving a suitable trade-off between approximation accuracy and computational and storage requirements depends largely on the characteristics of the specific problem at hand, including the number of features, the number of classes, the complexity of the density functions, the degree of overlap of the densities, the number of training samples available, etc. These factors must, in general, be studied experimentally (eg., through simulation) to determine whether it is preferable to use the exponential estimator or the polynomial approximation and in the latter case to arrive at the practical trade-offs required. In Section 6.5, some examples are given to illustrate the variation of approximation accuracy with the degree of the polynomial used, the magnitude of the smoothing parameters, and the location of the origin in

the feature space. The kinds of experiments discussed there can be used in practical situations for making the design decisions which have been described.

5.3 Calculating the Coefficients and Evaluating the Polynomials

The number of terms of degree n in a polynomial in X of dimension N is N^n; the total number of terms in a polynomial of degree $\bar{n}$ is

$$\sum_{n=0}^{\bar{n}} N^n .$$

Were it necessary to store or even to calculate this many terms, the polynomial approximation would not be very useful in practice except for problems of small dimensionality requiring only a low degree polynomial to provide sufficient approximation accuracy. Fortunately, both the storage and computational requirements can be substantially reduced.

Consider again a typical term of the polynomial expansion as given in (5.5):

$$\sum_{m=0}^{n^*} \frac{1}{m!(n - 2m)!} (X'M_iX)^{n^*-m} (X'\Sigma_iX) (X'P_i)^{n^{**}} .$$

For example, taking n = 5 and suppressing the pattern index i, this may be written

$$X'\left(\frac{1}{5!} M'XX'M + \frac{1}{3!} M'XX'\Sigma + \frac{1}{2!} \Sigma'XX'\Sigma \right) XX'P$$

or

$$\sum_{i=1}^{N} \cdots \sum_{r=1}^{N} \left(\frac{1}{5!} m_{ij} m_{k\ell} + \frac{1}{3!} m_{ij} \sigma_{k\ell} + \frac{1}{2!} \sigma_{ij} \sigma_{k\ell} \right) p_r x_i x_j x_k x_\ell x_r$$

Note that the matrices M and Σ are symmetric (see (5.4)). Let M^* be an N x N lower triangular matrix with diagonal elements equal to the diagonal elements of M and with subdiagonal elements equal to twice the corresponding elements of M; and similarly for Σ^* with respect to Σ. Then (5.6) becomes

$$\sum_{i=1}^{N} \sum_{j=1}^{i} \sum_{k=1}^{N} \sum_{\ell=1}^{k} \sum_{r=1}^{N} \left(\frac{1}{5!} m^*_{ij} m^*_{k\ell} + \frac{1}{3!} m^*_{ij} \sigma^*_{k\ell} + \frac{1}{2!} \sigma^*_{ij} \sigma^*_{k\ell} \right) p_r x_i x_j x_k x_\ell x_r \qquad (5.7)$$

Suppose N = 4. The number of terms associated with (5.6) is then $N^n = 4^5 = 1024$; the number of terms associated with (5.7) is $N^3(N-1)^2/4 = 144$, a considerable reduction. Still further economy can be had with respect to required storage, since all coefficients involving subscripts identical up to permutation may be added and stored as a single coefficient. Thus (5.7) may be written

$$\sum_{i=1}^{N} \sum_{j=1}^{i} \sum_{k=1}^{j} \sum_{\ell=1}^{k} \sum_{r=1}^{\ell} C_{ijk\ell r} x_i x_j x_k x_\ell x_r \qquad (5.8)$$

where the coefficients $C_{ijk\ell r}$ are appropriately defined. The number of coefficients stored for terms of degree n is then given by

$$\binom{N+n-1}{n}$$

which, for n = 5, N = 4 is just 56.

The classification procedure which requires, for each pattern classified, computation of all cross-products in the components of X (5.8), can be speeded up considerably if care is taken to compute successively higher degree cross-products using those of lower degree already calculated. If this is not done and the products are formed "from scratch" as needed,* a total of

$$n\binom{N+n-1}{n}$$

multiplications are required for the terms of degree n, or

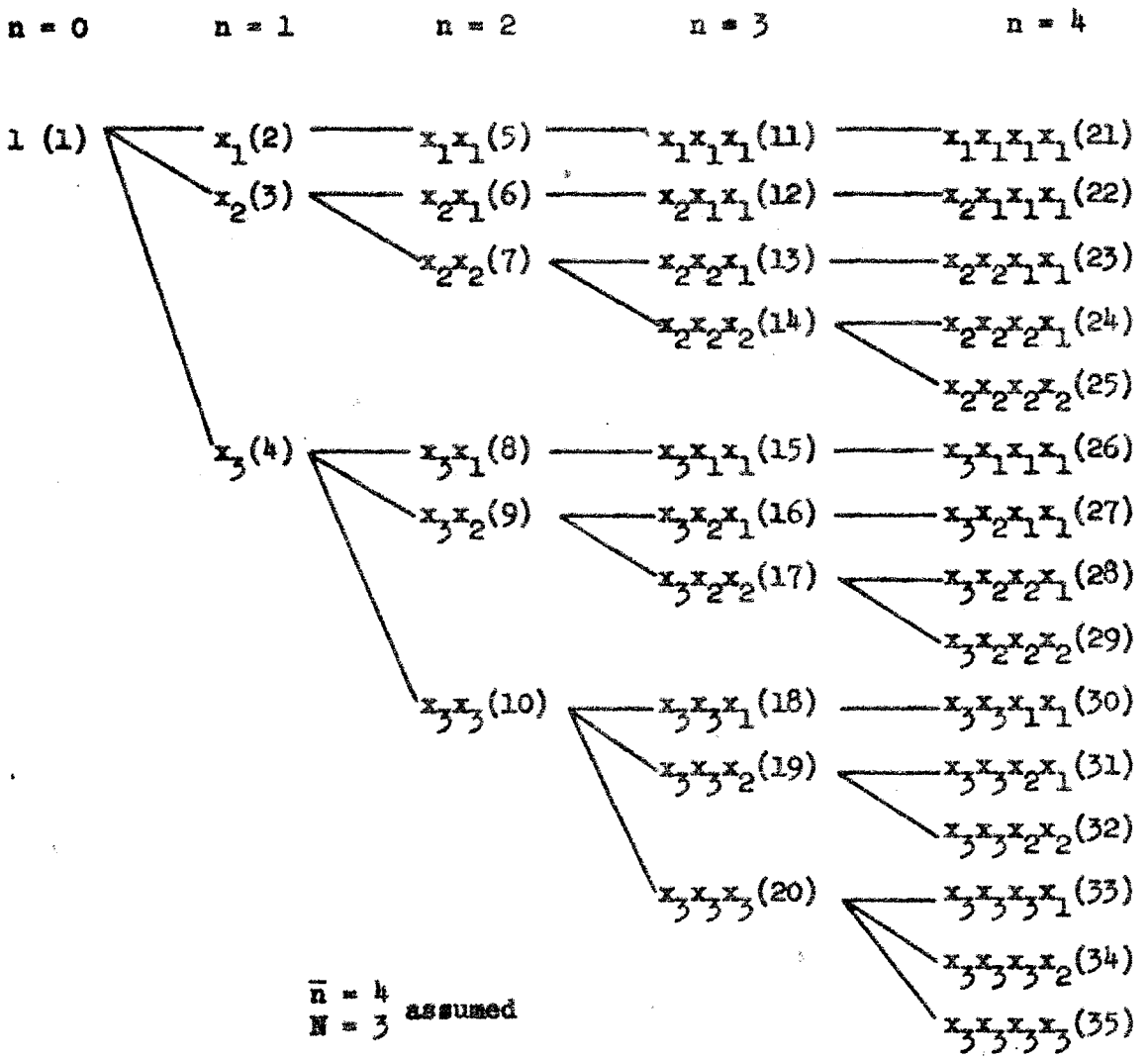
$$\sum_{n=0}^{\bar{n}} n\binom{N+n-1}{n} \tag{5.9}$$

for all terms up to degree $\bar{n}$ (the formulae include formation of the cross-products and multiplication by the corresponding coefficients). For n = 5, N = 4, (5.9) yields 504 multiplications. However, by calculating the cross products in the order suggested by Figure 5.1, only a single multiplication is required to form a new cross product from a previously computed result. In this case, only

$$2\sum_{n=1}^{\bar{n}} \binom{N+n-1}{n}$$

multiplications are required, which for n = 5, N = 4 amounts to 250 multiplications; and the saving becomes more dramatic with increasing n: n = 8, N = 4 yields 3168 multiplications the "long" way, 988 by the recommended way.

*Multiplication is assumed to be by far the most time consuming arithmetic operation used.



Numbers in parentheses indicate economical ordering of computations.

Figure 5.1 Suggested Ordering of Cross-Product Calculations

Obviously there is nothing very mysterious about the suggested ordering. In FORTRAN programming terms, the proper ordering of subscripts for terms of any given degree can be generated by a set of nested DO loops with each "inner DO" depending on the value of the preceding "outer DO" variable. The same idea is expressed by the iterated summation in (5.8) for terms of degree 5. However, because 1) it is necessary to reference, at each step, a previous result computed using a different sequence of subscripts (i.e., cross-products of lower degree), 2) the degree must vary from 1 up to the maximum desired level, and 3) the maximum level may not be known when the programming is accomplished, some ingenuity is needed to generate program code which is relatively efficient both from the point of view of the volume of code required and time necessary to execute the code during the training procedure (the most time consuming) or the recognition procedure (probably most critical in practice).

One additional detail deserves attention. In order to use the efficient coefficient storage, it is necessary to map the coefficient sequence associated with (5.7)

$$\{ijk \ldots \mid i = 1,2,\ldots,N;\ j = 1,2,\ldots,i;\ k = 1,2,\ldots,N;\ \ldots\}$$

onto the sequence associated with (5.8)

$$\{ijk \ldots \mid i = 1,2,\ldots,N;\ j = 1,2,\ldots,i;\ k = 1,2,\ldots,j;\ \ldots\}$$

so that every coefficient calculated by means of (5.7) may be added to the appropriate coefficient as defined by (5.8). It is easy enough to reorder the subscripts largest to smallest, but it is relatively difficult to then compute the pointer to the appropriate

coefficient memory cell, since combinatorial formulae akin to those already introduced must be used. Doing this repetitively for each coefficient and each training pattern adds enormously to the computational load. On the other hand, it is no more feasible to store and reference a complete table of subscript sets and pointers. Fortunately, a convenient compromise is possible which is now described.

Assume that the feature space is of dimension N and that the estimator is to be approximated by a polynomial of degree $\bar{n}$. Let α be the index which points to the location in the coefficient storage (which has been ordered according to (5.8)). Define a Coefficient Pointer Table (CPT) with elements TABLE(i,j) given by

$$\text{TABLE}(i,j) = \begin{cases} 0, & i < 1 \quad \text{or} \quad j < 1 \\ \\ \dfrac{i + j - 1}{i!\ (j-1)!}, & 1 \leq i \leq \bar{n},\ 1 \leq j \leq N + 1 \end{cases}$$

Then any term with subscripts which are a permutation of $i_1 i_2 \cdots i_r$ $(i_1 \geq i_2 \geq \cdots \geq i_r)$ is stored as part of the α^{th} coefficient, where

$$\alpha = 1 + \text{TABLE}(r-1,\ N+1) + \sum_{j=1}^{r} \text{TABLE}(r-j+1,\ i_j - 1)$$

(see Figure 5.2). Storing and referencing the CPT (which has size $\bar{n}$ x N+2) provides an economical trade-off between storing the entire set of coefficient indices and recalculating the indices each time they are needed.

TABLE (i,j)

	0	1	2	3	4	5	. . .	N + 1
1	0	1	2	3	4	5	. . .	N + 1
2	0	1	3	6	10	15	. . .	.
3	0	1	4	10	20	35	. . .	.
4	0	1	5	15	35	70	. . .	.
5	0	1	6	21	56	84	. . .	.
.	.	.	.	.	.	.		
.	.	.	.	.	.	.		
.	.	.	.	.	.	.		
$\bar{n}$	0	1	.	.	.	.		

Example: If N = 4, the coefficient index for C_{4311}, C_{4131}, C_{3141}, or C_{1143} is given by:

$$\alpha = 1 + 35 + 15 + 4 + 0$$

$$= 55$$

Figure 5.2 Coefficient Pointer Table (CPT)

Taken separately, the various suggestions outlined in this section provide a real (if in some cases modest) improvement in the computational and storage efficiencies of the polynomial approximation method. Taken together they may provide as much as an order of magnitude or more improvement in efficiency, depending on the problem dimensionality and the degree of the approximating polynomials to be used. This may make the difference between a possibly useful approach and a hopelessly impractical approach.

CHAPTER 6

EMPIRICAL INVESTIGATION

Chapters 3, 4, and 5 have left open a number of questions about the selection of certain parameters. In addition, it remains to be demonstrated that combination of the techniques proposed yields a workable pattern recognition system. These are the points addressed in this chapter.

6.1 The Advantage of "Data-Specific" Smoothing

It was mentioned in Section 3.1 that estimators have previously been investigated which are similar in form to the one discussed here but which use a single "universal" smoothing parameter (determined essentially by trial-and-error) for all pattern classes and features; i.e., in (3.2) the matrix S is taken as a scalar matrix and used for all pattern classes [8]. That the more general treatment proposed (for which each feature of each pattern class has its specific smoothing parameter) can be expected to produce improved pattern recognition performance is demonstrated by the results of the following experiment.

Thirty sets of artificial data were generated by a randomized procedure: A uniform random number generator was used to select the number of patterns per class and the means and standard deviations

of each feature for each class; given the randomly generated means and standard deviations, a Gaussian random number generator was used to produce the patterns.

Twenty sets of two-class three-dimensional data and ten sets of four-class three dimensional data were generated. The number of patterns per class was restricted to be between 30 and 60; the means were restricted to the range -25 to +25; and the standard deviations were restricted to the range 3 to 33. The allowed ranges for the means and standard deviations assured that the data would tend to be largely nonseparable (i.e., the classes would overlap in the usual sense), the situation in which the proposed methods are most advantageous.

Estimates of the underlying densities were formed in two ways and used for pattern recognition. In the first ("unnormalized") case, a single smoothing parameter was used, the value yielding the most accurate pattern classification determined by trial-and-error. In the second ("normalized") case, different smoothing parameters were selected for each feature of each class, the values taken proportional to the standard deviations of the respective features with the universal proportionality constant optimized by trial-and-error. The classification results are summarized in Table 6.1. The latter "data-specific" smoothing resulted in improved performance in 25 of 30 instances, degraded performance in 3, and unchanged performance in 2. The largest improvement observed (11.7 percent) was substantially greater than the largest degradation (1.8 percent).

Table 6.1 Comparison of Unnormalized and Normalized Smoothing

(a) 20 sets of artificial data, 2 classes, 3 features

Set	Percent Correct Unnormalized	Normalized*	Percent Improvement
1	83.1	94.8	11.7
2	82.8	83.9	1.1
3	99.1	99.1	-
4	80.0	85.3	5.3
5	76.1	79.5	3.4
6	85.7	87.1	1.4
7	83.0	81.2	-1.8
8	85.2	90.9	5.7
9	74.2	81.8	7.6
10	89.1	87.5	-1.6

Set	Percent Correct Unnormalized	Normalized*	Percent Improvement
11	89.8	91.8	2.0
12	79.6	80.6	1.0
13	83.6	85.3	1.7
14	88.1	89.1	1.0
15	100.0	100.0	-
16	94.0	95.2	1.2
17	97.5	98.8	1.3
18	75.6	82.2	6.6
19	84.5	90.5	6.0
20	67.1	70.7	3.6

(b) 10 sets of artificial data, 4 classes, 3 features

Set	Percent Correct Unnormalized*	Normalized*	Percent Improvement
1	73.4	74.0	0.6
2	71.4	73.5	2.1
3	65.7	68.5	2.8
4	83.4	87.4	4.0
5	66.4	65.0	-1.4
6	56.2	63.9	7.7
7	70.1	70.7	0.6
8	62.3	64.2	1.9
9	54.5	60.1	5.6
10	78.5	83.6	5.1

*by feature standard deviation

It may be concluded from this experiment that smoothing on an individual class/feature basis is generally superior to using a universal smoothing parameter. Although the average improvement observed is not large, there are cases in which it is substantial. In the few instances in which performance degrades, the loss is relatively insignificant.

Although these conclusions are based solely on sets of Gaussian data, it seems reasonable to expect comparable or even more favorable results for "data-specific" smoothing when the data are less well behaved. This was the observed tendency for the cases of real data to which the single parameter version was applied.

6.2 Selection of the Smoothing Constants k_1 and k_2

Since the immediate object in selecting values for k_1 and k_2 is to obtain a "good" estimate $\hat{p}(X)$ of the true or underlying probability density $p(X)$, a performance index based on the difference between $\hat{p}(X)$ and $p(X)$ could be used to evaluate candidate values. Ultimately the object is to achieve accurate pattern recognition, so alternatively one might choose to use recognition accuracy as the performance index. The latter approach is important when the underlying density is unknown and must be estimated from the training samples (usually the case in practice). However, some interesting results concerning the selection of k_1 and k_2 can be obtained by considering some artificial data with known underlying densities. These results are the subject of this section.

Integral square error, defined by

$$ISE = \int \left[\hat{p}(X) - p(X)\right]^2 dX \qquad (6.1)$$

where the integral is over the feature space, was selected as the criterion for optimizing the density estimate. Considering for simplicity the one-dimensional case, the density estimate based on samples from a population with unimodal density $p(x)$ is

$$\hat{p}(x) = \frac{1}{\sqrt{2\pi}\, r} \cdot \frac{1}{T} \sum_{i=1}^{T} \exp\left[-\frac{1}{2r^2}(x - y_i)^2\right]. \qquad (6.2)$$

The one-dimensional smoothing parameter r is, according to (3.10),

$$r = k_1 C(1) s T^{-1/k_2}. \qquad (6.3)$$

Given a set of observations with known underlying density, numerical techniques can be used on a digital computer to determine the value of r minimizing the ISE. Let r_1 denote the value of r found to minimize the ISE for a set of T_1 observations with sample variance s_1^2, and similarly for r_2, T_2 and s_2^2 (a second set of observations). Then

$$r_1 = k_1 C(1) s_1 T_1^{-1/k_2}$$

$$r_2 = k_1 C(1) s_2 T_2^{-1/k_2}.$$

Solving each of these equations for C(1) yields

$$C(1) = \frac{r_1 T_1^{1/k_2}}{k_1 s_1} = \frac{r_2 T_2^{1/k_2}}{k_1 s_2}$$

or

$$\left(\frac{T_1}{T_2}\right)^{1/k_2} = \frac{s_1\, r_2}{s_2\, r_1}.$$

Taking logarithms of both sides of the latter equation gives then, provided $T_1 \neq T_2$,

$$k_2 = \log\left(\frac{s_1\, r_2}{s_2\, r_1}\right) / \log\left(\frac{T_1}{T_2}\right). \tag{6.4}$$

The corresponding value of k_1 may then be computed from

$$k_1 = \frac{r_1\, T_1^{1/k_2}}{C(1)s_1} = \frac{r_2\, T_2^{1/k_2}}{C(1)s_2}. \tag{6.5}$$

This development suggests an experiment which should 1) indicate whether the form of (6.3) is indeed useful for estimating the smoothing parameter, and 2) yield values for k_1 and k_2. If in fact (6.3) gives a reasonable functional form for the smoothing parameter, then the resulting values of k_1 and k_2 should be independent of the selection of observation sets from the population (with the restriction that for any pair of sets selected, the sets must not contain the same number of observations).

Such an experiment was performed. To generate sets of observations from populations of known density, the following theorem was utilized.

Theorem ([10], Section 7.1.1): For any random variable x having a continuous cumulative distribution function F(x), the random variable $z = F(x)$ has the uniform distribution with density

$$f(z) = 1, \quad 0 \leq z \leq 1$$
$$= 0, \quad \text{otherwise.}$$

Stated a little differently, the theorem implies that if z is a random variable with the uniform probability density over the interval [0,1], then $x = F^{-1}(z)$ has the distribution F(x). For example, to generate the "triangular density" given by

$$p(x) = \frac{1}{b}\left(1 - \frac{|x|}{b}\right), \quad |x| \leq b \qquad (6.6)$$

$$= 0, \qquad \text{otherwise}$$

one has

$$z = F(x) = \begin{cases} 0, & x \leq -b \\ \frac{1}{2b^2}(x+b)^2, & -b < x \leq 0 \\ 1 - \frac{1}{2b^2}(x-b)^2, & 0 < x \leq b \\ 1, & x > b. \end{cases}$$

Inverting F(x), i.e., solving for x, yields

$$x = b\left[(2z)^{1/2} - 1\right], \qquad 0 \leq z \leq \frac{1}{2}$$

$$= b\left[1 - (2-2z)^{1/2}\right], \qquad \frac{1}{2} < z \leq 1.$$

Given uniformly distributed values of z, the corresponding values of x, computed from (6.7), have the desired triangular density.

In this manner, sets of observations from a population with arbitrary distribution F(x) can be generated by using a uniform random number generator, provided only that the inverse of F(x) can

be computed. However, note that if the observations are truly randomly generated, a certain amount of randomness will be present in the outcome of the experiment, thus requiring that many such experiments be performed and the results subjected to statistical analysis. Done in this manner, the overall experiment might quickly consume a large amount of computer time, even on a relatively fast machine. A possible alternative is the following: Rather than use z values actually produced by a uniform random number generator, one might use the "most likely" values for z; i.e., for T samples, take

$$z = \frac{1}{T}\left(i - \frac{1}{2}\right), \qquad i = 1, 2, \ldots, T.$$

This approach, which was used in performing the experiment, eliminates the randomness from the sample generation process and its effects from the computation of k_1 and k_2 which might obscure the kind of behavior of k_1 and k_2 which it is desired to observe.

Table 6.2(a) summarizes the experimental results for the triangular density (taking b = 1 in (6.6)). Over a considerable range in the number of observations used, the computed values for k_1 and k_2 are quite stable, as hoped.

A "cosinusoidal" density, for which the distribution function is also easily inverted, was tried. In this case,

$$p(x) = \frac{1}{2}\cos x, \qquad |x| \leq \pi/2$$

$$= 0, \qquad \text{otherwise.}$$

The distribution function is

Table 6.2 Determination of k_1 and k_2

(a) Triangular Density

T	s	r_{min}	ISE_{min}	k_1	k_2 for T_1 = T and T_2=40	80	160	320
20	.404	.095	$.27 \times 10^{-3}$					
40	.407	.067	$.95 \times 10^{-4}$	.30	2.0			
80	.408	.047	$.32 \times 10^{-4}$	.30	2.0	2.0		
160	.408	.034	$.10 \times 10^{-4}$	.29	2.0	2.0	2.0	
320	.408	.024	$.34 \times 10^{-5}$	.28	2.0	2.0	2.0	2.0

(b) Cosinusoidal Density

T	s	r_{min}	ISE_{min}	k_1	k_2 for T_1 = T and T_2=40	80	160	320
20	.678	.15	$.42 \times 10^{-4}$					
40	.682	.10	$.15 \times 10^{-4}$	.28	2.0			
80	.683	.076	$.52 \times 10^{-5}$	.28	2.0	2.0		
160	.683	.054	$.18 \times 10^{-5}$	.27	2.0	2.0	2.0	
320	.684	.039	$.62 \times 10^{-6}$	.25	2.0	2.0	2.0	2.1

(c) Normal Density

T	s	r_{min}	ISE_{min}	k_1	k_2 for T_1 = T and T_2=40	80	160	320
20	.969	.27	$.55 \times 10^{-4}$					
40	.984	.21	$.21 \times 10^{-4}$	.21	3.0			
80	.992	.17	$.87 \times 10^{-5}$	.20	3.1	3.1		
160	.996	.14	$.35 \times 10^{-5}$	.19	3.2	3.2	3.2	
320	.998	.11	$.14 \times 10^{-5}$	.19	3.2	3.2	3.2	3.3

T = no. samples; s = sample std. dev.; r_{min} = smoothing parameter at ISE_{min}; ISE_{min} = minimum integral square error.

$$z = F(x) = \begin{cases} 0, & x \leq -\pi/2 \\ \frac{1}{2}(1 + \sin x), & -\pi/2 < x \leq \pi/2 \\ 1, & x > \pi/2 \end{cases}$$

which inverted yields

$$x = \sin^{-1}(2z - 1), \qquad 0 \leq z \leq 1.$$

The experimental results for this case are given in Table 6.2(b). Again the variations in the computed values of k_1 and k_2 are small.

The distribution function for the normal or Gaussian density function cannot be analytically inverted. However, by computing the value of the distribution function for several values of x and using an interpolation process, the inversion can be numerically approximated to almost any desired accuracy. In this case, for the density function

$$p(x) = \frac{1}{\sqrt{2\pi}} \exp(-x^2/2), \qquad -\infty < x < \infty$$

the distribution function is

$$z = F(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} \exp(-\xi^2/2)\, d\xi$$

$$= \operatorname{erf}(x).$$

The error function (erf) is tabulated in many standard tables and can be numerically approximated and inverted on a digital computer. Table 6.2(c) summarizes the results for this experiment. Once again the computed values of k_1 and k_2 are quite stable.

Sweeping conclusions based solely on these three one-dimensional experiments should be avoided. However the results suggest an encouraging observation. At least for the one-dimensional case and the densities investigated, the relative constancy of the computed values for k_1 and k_2 indicates that (6.3) provides a useful functional form for the smoothing parameter. For a given density, values of k_1 and k_2 can be specified such that (6.3) gives an appropriate value of r. (It is conjectured that these values depend on the "smoothness" of the density, a conjecture which tends to be substantiated by the experimental results and by related theoretical efforts [11], [13].)

Unfortunately, extension of this experiment to multiple dimensions is impractical due to the multiple integrals which would have to be evaluated repeatedly. Therefore, investigation of dimensionality effects will necessarily be based on classification accuracy--discussed in Section 6.4--rather than integral square error.

Because of the qualitative observation that naturally occurring data often exhibit density functions which tend to be most like the cosinusoidal density discussed above (of finite extent but smoother than the triangular density), the values $k_1 = 0.3$ and $k_2 = 2.0$ have been used in the pattern recognition experiments of Sections 6.4 and 6.5. The results of those sections bear further on the question of the suitability of the smoothing parameter selection.

6.3 A One-Dimensional Experiment for Evaluating the Mode Discrimination Threshold

To observe the effects of mode separation on smoothing parameter computation and density estimation error, artificial one-dimensional data sets from populations with known density functions were generated, each data set consisting of the union of two subsets identically distributed except for means symmetrically displaced positively and negatively from zero (see Figure 6.1). The proposed exponential estimator was applied to each data set in two ways. First, the artificial data set was considered to be unimodal and the smoothing was computed based on the overall sample standard deviation. Second, the data set was sectioned at the origin into two "modes" and the smoothing was based on the sample standard deviations of the modes. The integral square error (ISE) was computed for each estimate of the density function.

Let ISE1 be the error resulting from considering the data to be unimodal; let ISE2 be the error resulting from considering the data to be bimodal. Define the normalized separation of the modes to be the distance between the mode centers (as defined in Chapter 4) divided by twice the radius of the ellipsoid of concentration. The normalized separation corresponds to the separability measure proposed in Chapter 4. Figure 6.2 shows, for two of the artificially generated densities discussed in the preceding section, the behavior of the ratio ISE1/ISE2 as a function of the normalized separation of the modes. In each case the ratio rises sharply beyond a separation

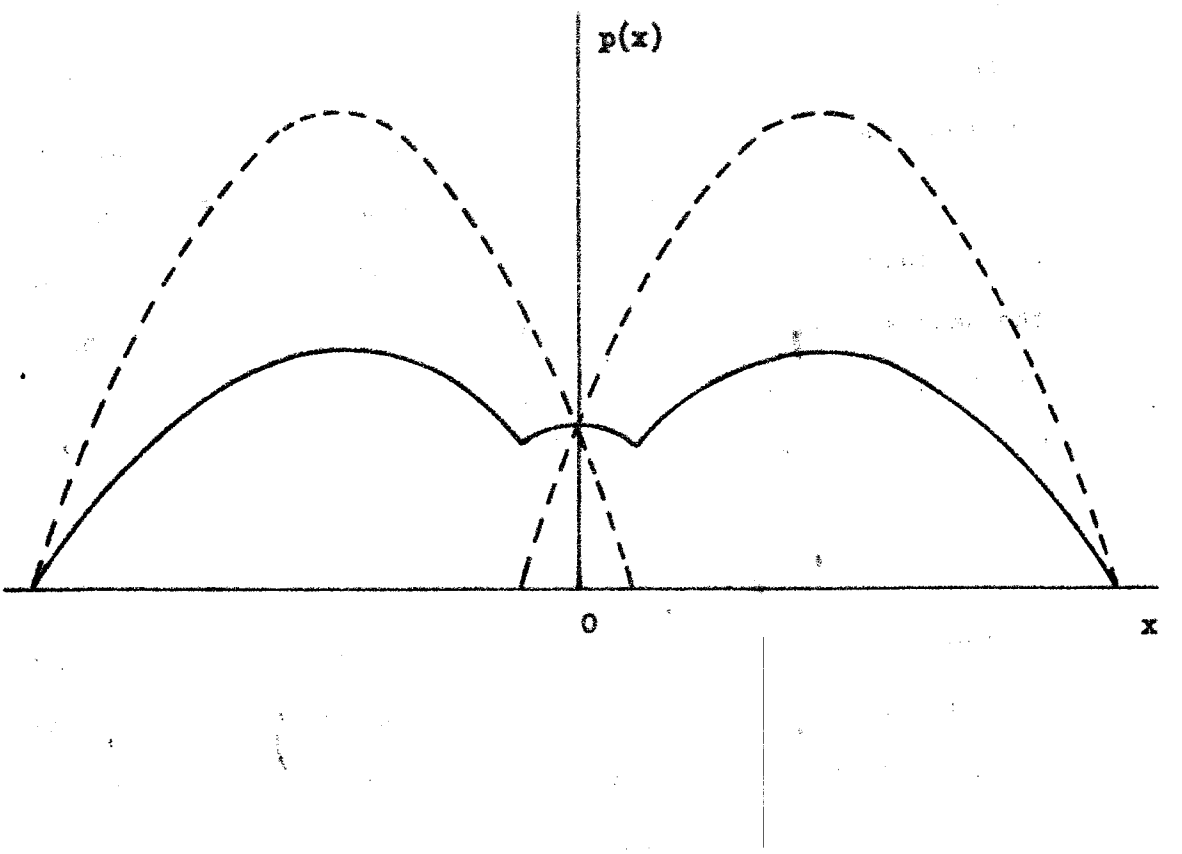
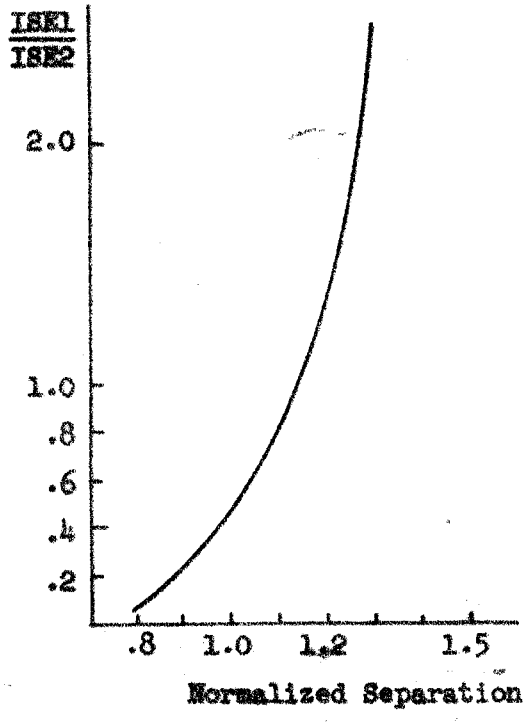
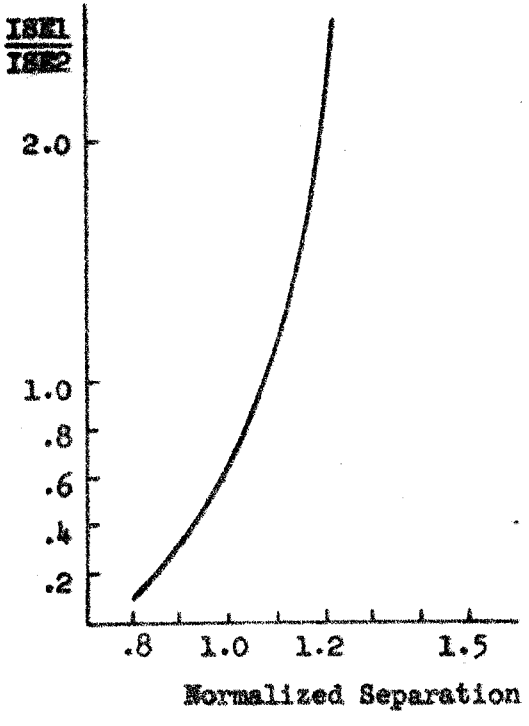


Figure 6.1 Synthesis of a Multimodal Density Function



(a) Cosine Density



(b) Triangular Density

Figure 6.2 Relative Estimation Error as a Function of Normalized Mode Separation

of about 0.8 and exceeds unity beyond a separation slightly greater than 1.0. A ratio greater than unity indicates that the error resulting from the unimodal assumption is greater than the error resulting from the bimodal assumption. Therefore, for the density functions considered in this experiment, the optimal separability threshold γ_t (see (4.7)) is approximately unity. Beyond this (normalized) separation, the intramodal standard deviation apparently provides a better measure of local dispersion than does the overall standard deviation.

Since the definition of ellipsoid of concentration contains an inherent compensation for changing dimensionality, it seems reasonable to expect that a threshold near unity may also be appropriate for dimensionalities greater than one. One would like to conjecture that the unity threshold can also be applied to more general density forms such as those likely to occur in real data. Computational difficulties (principally the evaluation of multiple integrals as noted in the preceding section) again discourage the extension of the experiment to more general situations. However, the experiments in the following section, using real data, tend to support the conjecture that $\gamma_t = 1$ is a reasonable choice.

6.4 Mode Discrimination and Pattern Recognition Experiments

An extensive series of experiments were conducted involving several sets of real data in order to test the effectiveness and practical utility of the methods formulated in Chapters 3 and 4.

Specifically, the goals of these experiments included:

1. Further investigation of the mode separation threshold γ_t for specifying "significantly distinct" modes or clusters of data.

2. Overall evaluation of the proposed mode discrimination algorithm, especially its effectiveness in partitioning data to aid in proper smoothing parameter determination.

3. Further verification that the proposed expression for computing smoothing parameters (3.10) leads to accurate probability density estimation for real data involving a range of dimensionalities and numbers of training samples.

4. Overall evaluation of the probability density estimator as a tool for application to pattern recognition.

The results tabulated in this section were obtained as follows:

1. A subset of the data (not necessarily a proper subset) was selected as training data for preprocessing.

2. The mode discrimination algorithm proposed in Chapter 4 was applied to the training data and smoothing parameters were computed based on the results. In addition to the proposed criterion for measuring mode distinctness, all pairwise combinations of data features were plotted by computer and examined visually. Whenever the visual analysis raised any doubt about the modal analysis results, all reasonable possibilities were tested in the following pattern recognition step.

3. The proposed density estimator was used for classifying the entire set of data (including the training data). Several

classifications of the data were performed, each classification using as smoothing parameters a different scalar multiple (smoothing parameter multiplier) of the smoothing matrix determined in step 2.

4. For comparison, the data were classified by the Gaussian maximum likelihood criterion, using the training samples to estimate the class mean vectors and covariance matrices.

Equal a priori class probabilities were assumed for all of the pattern recognition experiments.

Three distinct sets of data were used (identified as data set "A", data set "B", and data set "360"), from which 22 sets of experiments were composed, 53 experiments in all.* Each set of experiments differed in some combination of the data set, the number of features, and the number of data samples used for training. Each experiment within a set differed in the number of modes used for one or more classes in the classifications. The results of the experiments, listed in Tables 6.3, 6.4, and 6.5 are interpreted below in terms of the experimental goals.

---

*The original data consisted of reflectance measurements on plant leaves, made by means of a Beckman DK-2A spectroreflectometer over the spectral range 0.26 μ to 2.60 μ in increments of 0.01 μ (see "Remote multispectral sensing in agriculture," vol. 2, Lab. for Agricultural Remote Sensing, Purdue University, Lafayette, Ind., July 1967.). For the experiments described here, the data were preprocessed by averaging over nine segments of the spectral range in order to simulate the output of an airborne multispectral scanner. Each of the nine pattern features corresponds to one of the spectral segments, which were typically (in microns): .50 - .52; .52 - .55; .55 - .58; .58 - .62; .62 - .66; .66 - .72; .72 - .80; .80 - 1.0; 1.5 - 1.8.

Table 6.3 Results for "A" Data[+]

Features	No. Training Samples/Class		No. Modes in Class		δ_{opt} *	No. Correct at δ_{opt}		Total % Correct at δ_{opt}	Mode Separation Indices	
	1	2	1	2		1	2		Y_1	Y_2
4,8	97	108	2	2	1.25	85	97	88.8	1.14	1.88
4,8	97	108	1	1	1.00	86	95	88.3	-	-
2,4,6,8	97	108	2	2	0.50	94	102	95.6	1.34	2.29
2,4,6,8	97	108	1	1	0.40	91	102	94.1	-	-
4,8	24	26	2	2	1.25	81	95	85.8	1.10	0.71
4,8	24	26	2	1	1.75	81	96	86.3	1.10	-
4,8	24	26	1	1	1.00	84	95	87.3	-	-
2,4,6,8	24	26	2	2	1.00	91	94	90.2	1.34	1.06
2,4,6,8	24	26	2	1	0.60	88	95	89.3	1.34	-
2,4,6,8	24	26	1	1	0.40	86	94	87.8	-	-
4,8	24	80	2	2	1.50	84	99	89.3	1.10	1.74
4,8	24	80	1	2	1.25	82	98	87.8	-	1.74
4,8	24	80	1	1	1.50	82	100	88.8	-	-
4,8	72	26	2	2	1.00	83	95	86.8	1.07	0.71
4,8	72	26	2	1	1.75	82	96	86.8	1.07	-
4,8	72	26	1	1	1.25	82	93	85.4	-	-
2,4,6,8	24	80	2	2	0.60	86	102	91.7	1.34	2.09
2,4,6,8	24	80	1	1	0.50	82	103	90.2	-	-

Table 6.3, cont.

Features	No. Training Samples/Class 1	2	No. Modes in Class 1	2	δ_{opt} *	No. Correct at δ_{opt} 1	2	Total % Correct at δ_{opt}	Mode Separation Indices γ_1	γ_2
2,4,6,8	72	26	2	2	0.90	89	96	90.2	1.25	1.06
2,4,6,8	72	26	2	1	0.20	91	93	89.8	1.25	-
2,4,6,8	72	26	1	1	1.00	88	96	89.8	-	-
1 thru 9	97	108	2	2	0.50	93	104	96.1	1.40	2.56
1 thru 9	97	108	1	1	0.40	92	105	96.1	-	-

+ Class 1: 97 samples of soybeans; class 2: 108 samples of corn.

* Smoothing parameter multiplier giving best recognition results (see text).

Table 6.4 Results for "B" Data[+]

Features	No. Training Samples/Class 1	2	No. Modes in Class 1	2	δ_{opt} *	No. Correct at δ_{opt} 1	2	Total % Correct at δ_{opt}	Mode Separation Indices γ_1	γ_2
4,8	100	100	2	2	1.00	88	86	87.0	0.75	0.81
4,8	100	100	1	1	1.05	87	87	87.0	-	-
2,4,6,8	100	100	2	2	0.75	89	82	85.5	0.81	0.76
2,4,6,8	100	100	1	1	1.00	91	80	85.5	-	-
4,8	24	24	2	2	1.00	88	85	86.5	0.74	0.73
4,8	24	24	1	1	0.70	89	85	87.0	-	-
2,4,6,8	24	24	2	2	1.00	87	85	86.0	0.69	0.53
2,4,6,8	24	24	1	1	0.40	87	86	86.5	-	-
4,8	24	74	2	2	0.75	80	90	85.0	0.74	0.60
4,8	24	74	1	1	1.50	81	89	85.0	-	-
4,8	74	24	2	2	1.00	92	83	87.5	0.70	0.73
4,8	74	24	1	1	0.75	92	81	86.5	-	-
2,4,6,8	24	74	2	2	2.00	85	87	86.0	0.69	0.67
2,4,6,8	24	74	1	1	0.70	81	90	85.5	-	-
2,4,6,8	74	24	2	2	0.80	94	76	85.0	0.77	0.53
2,4,6,8	74	24	1	1	0.80	95	75	85.0	-	-

Table 6.4, cont.

Features	No. Training Samples/Class 1	2	No. Modes in Class 1	2	δ_{opt} *	No. Correct at δ_{opt} 1	2	Total % Correct at δ_{opt}	Mode Separation Indices γ_1	γ_2
1 thru 9	100	100	2	2	.30	84	86	85.0	0.81	0.70
1 thru 9	100	100	1	1	.35	90	81	85.5	-	-
1 thru 9	24	24	2	2	.60	86	87	86.5	0.80	0.59
1 thru 9	24	24	1	1	.50	90	84	87.0	-	-

+ Class 1: 100 samples of oats; class 2: 100 samples of wheat.

* Smoothing parameter multiplier giving best recognition results (see text).

Table 6.5 Results for "360" Data[+]

Features	Number of Training Samples/Class				No. Modes in Class				δ_{opt} [*]	Number Correct at δ_{opt}				Total % Correct at δ_{opt}	Mode Separation Indices[$]			
	1	2	3	4	1	2	3	4		1	2	3	4		γ_1	γ_2	γ_3	γ_4
8,8	45	106	105	104	2	2	2	3	1.00	42	96	83	103	90.0	.97	.87	1.62	1.23
2,8	45	106	105	104	1	1	2	3	1.00	43	95	82	103	89.7	-	-	1.62	1.23
2,8	45	106	105	104	1	1	1	1	0.75	43	99	77	98	88.1	-	-	-	-
2,4,6,8	45	106	105	104	2	1	2	3	1.00	43	106	94	100	95.3	.96	-	1.35	1.29
2,4,6,8	45	106	105	104	1	1	2	3	1.00	44	105	97	98	95.6	-	-	1.35	1.29
2,4,6,8	45	106	105	104	1	1	1	1	0.50	42	105	96	92	93.1	-	-	-	-
2,8	33	69	26	51	2	2	2	3	1.25	42	97	85	95	88.6	.88	.84	1.38	1.17
2,8	33	69	26	51	2	1	2	3	1.10	41	97	85	94	88.1	.88	-	1.38	1.17
2,8	33	69	26	51	1	1	2	3	1.10	42	96	85	94	88.1	-	-	1.38	1.17
2,8	33	69	26	51	1	1	1	1	0.80	41	102	79	90	86.7	-	-	-	-

+ Class 1: 45 samples of soybeans; class 2: 106 samples of corn; class 3: 105 samples of oats; class 4: 104 samples of wheat.

* Smoothing parameter multiplier giving best recognition results (see text).

$ Where a class has more than 2 modes, the value given is the minimum intermode index.

6.4.1 Mode Separation Threshold

Tables 6.6 and 6.7 summarize the results in terms of the relationships observed between mode separation and classification performance. In these tables, a combination of the outcomes of two experiments within a set was counted as "admissible" if applying the mode discrimination technique with the specified threshold indicated subdivision of data modes when and only when such subdivision yielded improved pattern recognition performance; otherwise the combination was counted as "inadmissible". (In a few cases, combinations involved more than one difference in mode configuration with observed separation indices straddling the threshold value. Such cases were counted "not applicable".)

It is clear from Tables 6.3, 6.4, and 6.5 that a threshold value somewhere near $\gamma_t = 1$ is suitable. Table 6.6 gives a more detailed breakdown for values near unity. Although the classification results were fairly insensitive to the precise threshold value, $\gamma_t = 1.00$ produced the best performance by a narrow margin. These experimental results support the outcome of the artificial data experiment reported earlier. As shown in Table 6.7, applying $\gamma_t = 1.00$ to each set of experiments (rather than to all combinations of experiments within sets) indicates that modal analysis produced admissible results for 82 percent of the data sets. (Twelve of the data sets were analyzed as containing one or more multimodal classes, 10 as containing exclusively unimodal classes based on the mode discrimination technique.)

Table 6.6 Mode Distinctness Thresholding Near γ_t=1

Threshold γ_t	Cases Admissible / Inadmissible (Not Applicable) "A" Data	"B" Data	"360" Data	Overall	Percent Admissible*
0.80	10/4 (0)	5/2 (3)	5/2 (0)	20/8 (3)	71
0.85	10/4 (0)	8/2 (0)	4/3 (0)	22/9 (0)	71
0.90	10/4 (0)	8/2 (0)	4/2 (1)	22/8 (1)	73
1.00	10/4 (0)	8/2 (0)	5/2 (0)	23/8 (0)	74
1.10	8/6 (0)	8/2 (0)	5/2 (0)	21/10 (0)	68
1.15	7/6 (1)	8/2 (0)	5/2 (0)	20/10 (1)	67

* (No. Admissible/No. Applicable) x 100

Table 6.7 Overall Performance for Mode Discrimination (γ_t=1)

Sets Admissible / Inadmissible "A" Data	"B" Data	"360" Data	Overall	Percent Admissible
7/2	8/2	3/0	18/4	82

6.4.2 Overall Evaluation of the Mode Discrimination Technique

The proposed mode discrimination technique has proven admirably suited to the task for which it was intended. The experimental results indicate that it does indeed structure the data so as to avoid gross smoothing errors due to variances exaggerated by multimodal data structure. From a practical viewpoint, the algorithm is quite fast due to the relatively simple computations involved and converges rapidly to the final result.

It might be mentioned here that graphical computer output can be of considerable value in this type of pattern processing. The program used in the experimental system produces plots of the data by pairs of features. These plots help reassure the user that a desirable data structuring (modal analysis) is in fact achieved. Although it may not be practical to print all pairwise feature plots when the number of features is large, a sampling of such plots is still helpful; and the availability of a CRT graphics facility would make possible the on-line examination of a very large number of such displays.

6.4.3 Pattern Recognition Accuracy and Smoothing Parameter Selection

Based on pattern recognition accuracy as a standard, the proposed density estimation method may be considered successful. As shown in Table 6.8, the recognition accuracy achieved by the exponential estimator (with prior modal analysis) was better for 13 of 22 sets of data than the accuracy produced by the parametric assumption of Gaussianly distributed data. The performance for the two

Table 6.8 Performance Comparison for the Exponential Estimator and a Parametric Estimator

Data Set	Features	Number of Training Samples/Class	Nonparametric Results (% Correct)	Parametric Results (% Correct)	Nonparametric Advantage (%)
A	4,8	97,108	88.8	88.3	+0.5
A	2,4,6,8	97,108	95.6	95.1	+0.5
A	4,8	24,26	87.3	87.3	-
A	2,4,6,8	24,26	90.2	93.2	-3.0
A	4,8	24,80	89.3	87.8	+1.5
A	4,8	72,26	86.8	88.3	-1.5
A	2,4,6,8	24,80	91.7	94.6	-1.9
A	2,4,6,8	72,26	90.2	93.2	-3.0
A	1-9	97,108	96.1	99.0	-2.9
B	4,8	100,100	87.0	85.0	+2.0
B	2,4,6,8	100,100	85.5	84.5	+1.0
B	4,8	24,24	87.0	85.0	+2.0
B	2,4,6,8	24,24	86.5	86.5	-
B	4,8	24,27	85.0	85.5	-0.5
B	4,8	74,24	87.5	86.0	+1.5
B	2,4,6,8	24,74	86.0	85.5	+0.5
B	2,4,6,8	74,24	85.0	81.5	+3.5
B	1-9	100,100	85.5	91.5	-6.0
B	1-9	24,24	87.0	86.5	+0.5
360	2,8	45,106,105,104	90.0	84.7	+5.3
360	2,4,6,8	45,106,105,104	95.6	92.2	+3.4
360	2,8	33,69,26,51	88.6	84.7	+3.9

approaches was the same for two additional sets and better for the parametric assumption for the remaining 7 sets. Interestingly, five of the latter seven cases involved the "A" data, which happened to have the most highly correlated features. The Gaussian assumption has the ability to use correlation information, inherent in the computation of the covariance matrix. The smoothing parameters utilized by the proposed density estimation technique do not account for correlation of features, and the experimental results suggest the value of extending the method to accomplish this. Such an extension is theoretically straightforward (a generalized form of smoothing matrix is required), but of course would entail increasing the computational and storage requirements of the method.

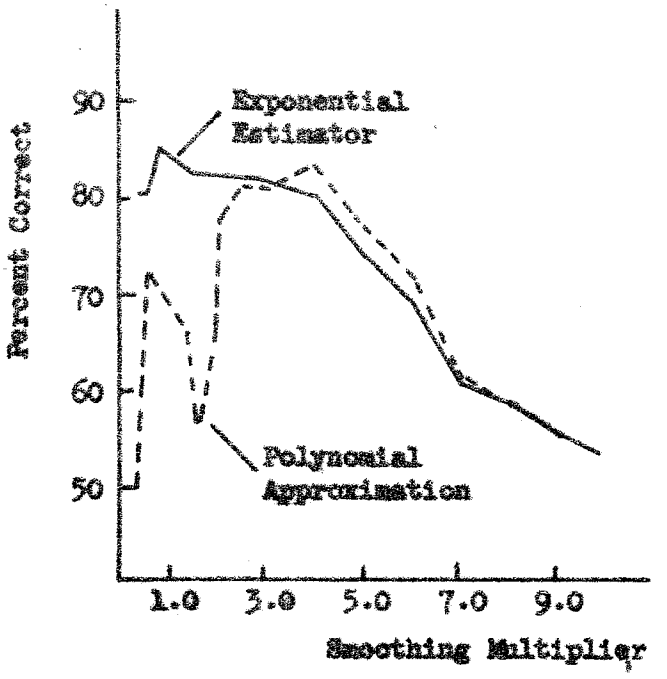
Apparently the proposed functional form for the smoothing parameters does provide a useful estimate of optimal smoothing, as evidenced by the recognition results and the relatively narrow range near unity observed for the optimal smoothing parameter multiplier. The 53 experiments involved ranges of from 2 to 9 features and 24 to 106 patterns per class, yet there does not seem to be any consistent trend in the optimal smoothing parameter multipliers that would suggest an essential insufficiency in the proposed functional form. Also, the values for k_1 and k_2 selected as a result of the artificial data experiment described in Section 6.2 appear to have been adequate for all of the real data studied.

6.5 Performance of the Polynomial Approximation

Figure 6.3 shows a typical plot of recognition performance for both the exponential estimator and the polynomial approximation as a function of the smoothing multiplier δ (the approximation is a 7th degree polynomial). As seen earlier, the performance of the exponential estimator tops out near δ = 1. In the case shown, the polynomial could not accurately approximate all of the detail contained in the exponential estimator for δ = 1. But by allowing a somewhat greater degree of smoothing, the performance of the approximation for this case could be brought up to a level fairly close to that achieved by the exponential estimator. The sacrifice in accuracy is compensated by the improvement in classifier speed and reduced storage requirements. The exponential estimator required 100 storage locations for the training patterns (2 classes, 2 features, 25 training samples) and 9.7 seconds to classify 150 patterns. The polynomial approximation required 72 storage locations (36 coefficients for each of 2 classes) and only 3.4 seconds to classify the same 150 patterns (comparable figures for 200 training patterns, 200 patterns classified are: 400 storage locations, 41.3 seconds versus 72 storage locations, 4.4 seconds).*

Notice that for a certain range of smoothing parameters, the performance of the polynomial approximation appears to be better than that of the exponential estimator. This anomaly is explained by

*These figures are of course machine and program dependent. They are intended only to give some idea of the relative efficiences involved. The computer used in this experiment was an IBM System/360 Model 44 and the classifiers were programmed in FORTRAN.



Data Set "A"; Features 2 and 8; 24 Training Samples for Class 1, 26 Training Samples for Class 2 (Training Samples Not Included in Test Set)

Figure 6.3 Performance of the Exponential Estimator and a Polynomial Approximation

looking at a plot of the data (Figure 6.4) and checking the magnitudes of the density estimates as computed by the two methods. It happens that the "superior" performance of the approximation results from the unreliable behavior of the approximation at large distances from the origin of the feature space. Classification by the two methods was almost identical near the origin, but the approximation coincidentally classified a number of "outliers" in one class correctly (there were no training samples from either class in the region in which this occurred), thereby accounting for most of the apparent difference in performance. Assuming the various parameters have been suitably chosen (polynomial degree, smoothing multiplier, feature space origin), this sort of behavior can be expected to occur only in regions associated with very low probabilities and thereby should have relatively insignificant effect on the overall performance of the classifier.

The effect of origin placement is further demonstrated by the results shown in Table 6.9. Specht [8] suggested that the origin should be shifted to the average of the means of the classes. It is clear, however, that the relative dispersions of the classes must be taken into account if the origin is to be located near the decision boundary. Experience has shown that further advantages are obtained by explicitly taking account of the multimodal structure of the data if it is available. Therefore, the classifier has been programmed to shift the origin to a dispersion-weighted mean of modes, computed as follows. Let $m(i,j,k)$ and $s(i,j,k)$ be respectively the sample mean

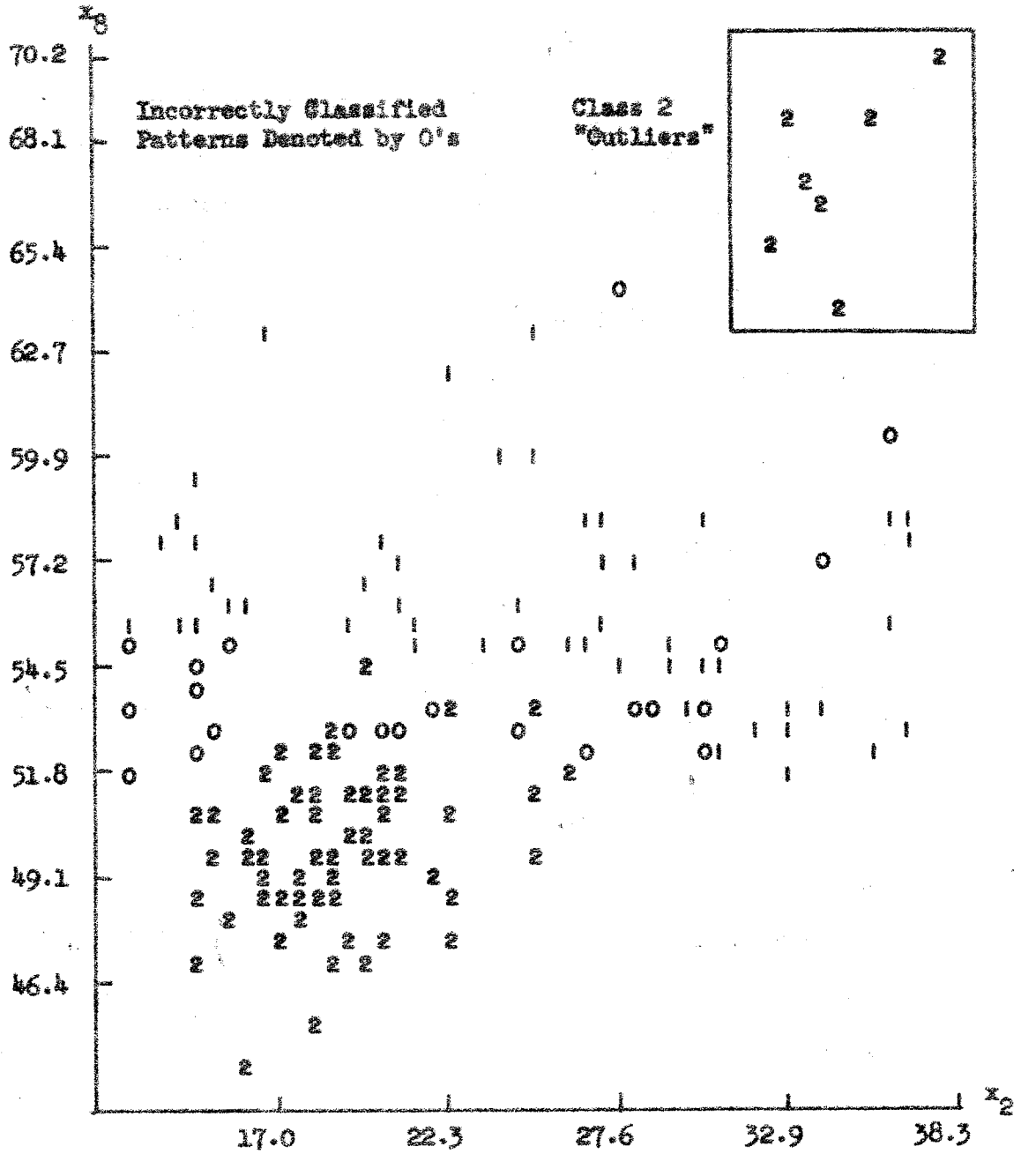


Figure 6.4 Anomalous Classification of "Outliers"

Table 6.9 Effects of Origin Location*

Smoothing Multiplier	Performance (% Correct) for Exponential Estimator	Approximation with Computed Origin Shift	Approximation with Visually Adjusted Origin
0.75	86.5	38.0	68.5
1.00	87.0	54.5	70.5
1.25	86.0	56.5	73.0
1.50	85.0	64.5	74.5
1.75	85.0	72.0	80.5
2.00	85.5	74.0	84.5
2.25	85.5	78.0	85.0
2.50	86.5	79.5	85.5
2.75	86.5	82.0	82.0
3.00	85.0	82.5	80.0
3.25	85.0	83.0	79.5
3.50	82.5	82.5	78.5
3.75	80.5	80.0	76.0
4.00	78.5	78.5	74.5
4.25	78.0	78.5	74.5
4.50	79.0	79.0	76.0
4.75	76.0	76.0	74.0
5.00	71.5	72.5	70.0

* on data set "B"; features 4,8; 100 training samples per class; 1 mode per class; training patterns included in test set.

and sample standard deviation of the k^{th} component of the j^{th} mode center associated with the i^{th} pattern class. The k^{th} component of the shifted origin is given by (assume K classes, n_i modes in the i^{th} class):

$$o_k = \frac{\sum_{i=1}^{K} \sum_{j=1}^{n_i} s(i,j,k)\, m(i,j,k)}{\sum_{i=1}^{K} \sum_{j=1}^{n_i} s(i,j,k)} . \qquad (6.8)$$

Equation (6.8) has produced nearly optimal results in most of the cases tried, but further improvement is sometimes possible. Figure 6.5 shows the computed origin and a visually adjusted origin superimposed on a plot of the results obtained for the visually adjusted case. The visually selected origin was simply chosen to lie in a region both densely occupied by patterns and very near the apparent decision boundary resulting from the exponential estimator. As a result of the visual adjustment, the quality of the approximation in this region improved and, with the use of a smaller smoothing multiplier, the obtainable recognition performance improved by a few percent.

The sort of origin adjustment performed above, based on the actual classification results, could be done automatically by a suitable algorithm designed to seek regions of maximum error density.

Finally, Table 6.10 illustrates the behavior of the polynomial approximation with variations in the degree of the polynomial used. The table illustrates several interesting points in addition to those

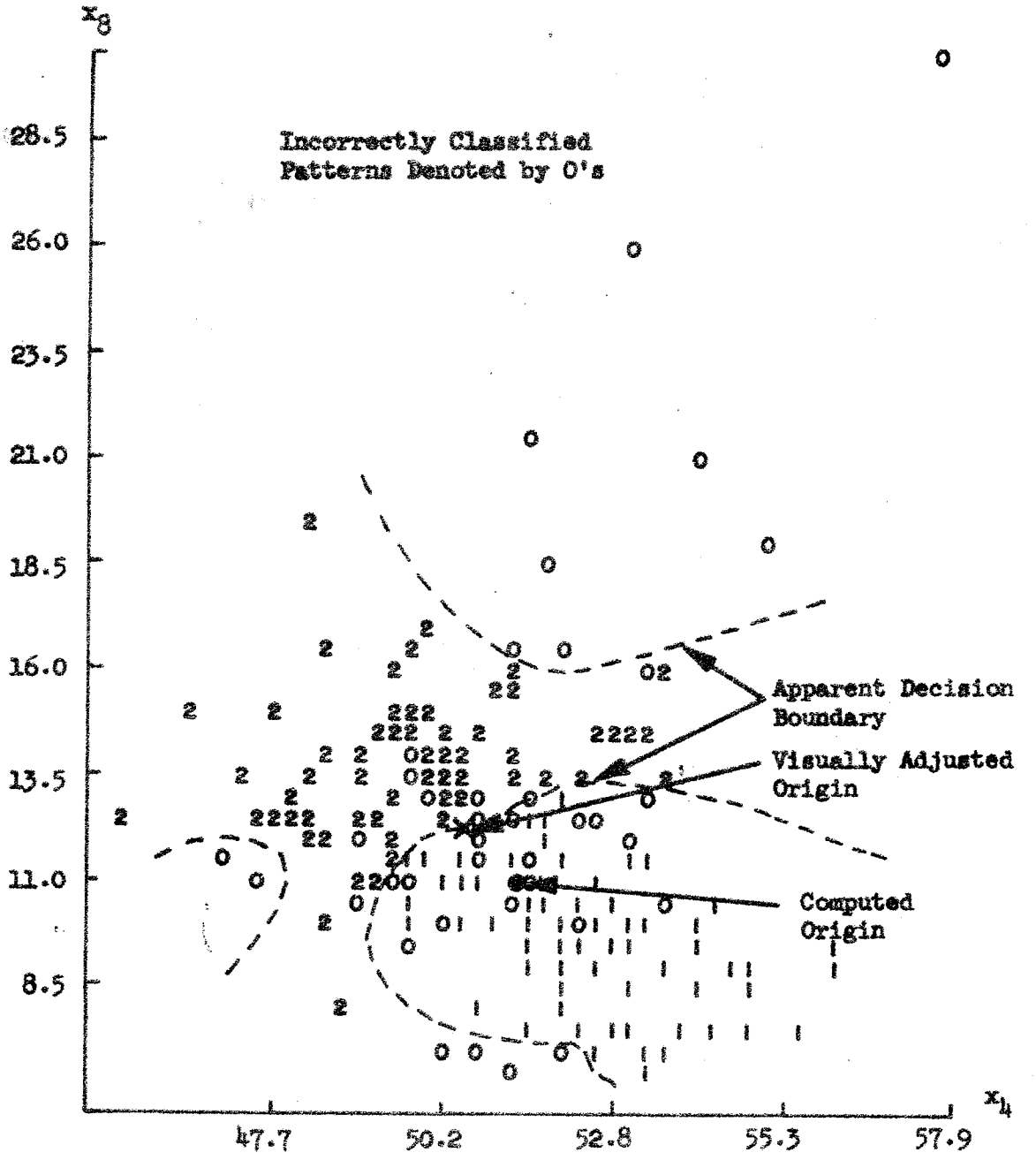


Figure 6.5 Visual Origin Adjustment

Table 6.10 Approximation Performance as a Function of Polynomial Degree*

Smoothing Multiplier	Performance (% Correct) for Exponential Estimator	Polynomial Approximation of Degree 7	6	5	4	3	2
0.5	80.6						
1.0	84.2	69.0	71.0	76.8	46.4	37.4	70.3
2.0	83.2	77.4	73.5	77.4	76.8	74.2	82.6
3.0	81.9	81.3	81.9	78.7	74.2	80.6	82.6
4.0	80.6	83.2	80.6	80.0	78.7	81.9	81.9
5.0	74.2	76.8	74.2	74.2	76.1	78.1	78.1
6.0	69.7	72.3	69.7	69.7	71.6	72.9	71.6
7.0	60.6	61.3	61.3	60.6	63.2	65.2	65.2
8.0	58.1	58.1	58.1	58.1	61.3	61.3	60.6
9.0	55.5	55.5	55.5	55.5	57.4	55.5	58.1
10.0	53.5	53.5	53.5	53.5	54.2	52.9	55.5

* on data set "A"; features 2,8; 24 training patterns for class 1, 26 for class 2; 1 mode per class; test set exclusive of training patterns.

already mentioned above. Notice first that the polynomials of degree 5, 6, and 7 converge at such a rate with increasing smoothing that the pattern recognition results for $\delta \geq 8.0$ are identical with the results produced by the exponential estimator. For polynomial degree smaller than 5, even a smoothing multiplier equal to 10 is insufficient to produce identical results. Surprising perhaps is the relatively good performance achieved by the lower degree polynomials for smaller values of δ (the upper right portion of the table). This does not indicate that good approximations to the exponential estimator have been achieved but rather that some lower order decision surfaces have been found capable of giving the indicated levels of performance. It is not altogether a coincidence that the optimum performance for the 2nd degree polynomial is 82.6 percent: This is exactly the performance produced for this data by the parametric classifier assuming Gaussianly distributed data, which of course produces a decision surface of degree 2. But the table also shows that the poorer approximation cannot be depended on to achieve good results, particularly for small smoothing parameters: Notice the fluctuating performance in the extreme upper right portion of the table, which results from the unpredictable behavior of the approximation when the approximation error is large.

6.6 Summary

It has been demonstrated that the proposed nonparametric approach for estimating probability density functions based on training samples can be profitably applied to pattern recognition problems involving

complex decision boundaries. The mode discrimination algorithm has proven a successful and efficient method for partitioning multimodal data so as to obtain the proper smoothing parameters for the exponential estimator. Investigation of the effects of mode separation on estimation error and pattern recognition performance has shown that $\gamma_t = 1.0$ is a nearly optimal mode distinctness threshold for a majority of cases.

The experimental evidence further indicates that the functional form suggested for the data-specific smoothing parameters can be expected to produce a useful estimate of the optimal smoothing for real data. The parameters k_1 and k_2 associated with the functional form have been estimated by means of experiments with one-dimensional artificial data and the resulting values have proven satisfactory for use with real multidimensional data.

The exponential estimator cannot be used indiscriminantly, however. Its performance on data with highly correlated features is certainly less than optimal, a problem which might be alleviated by further generalizing the smoothing procedure. The most serious disadvantage of the exponential estimator, however, is the requirement that the entire set of training samples be saved and used in computations for each classification to be performed. The resulting memory requirements and computation times may be intolerably excessive in case of large training sets. This difficulty can be circumvented to an extent by using a polynomial approximation of the estimator. Still, considerable effort is required in implementing the

approximation to ensure minimal memory and computation requirements and maximal pattern recognition accuracy.

CHAPTER 7

THE LINGUISTIC APPROACH TO PATTERN RECOGNITION: INTRODUCTION

As noted in Chapter 1, the goal of the linguistic approach to pattern recognition is to make effective use of the methodologies of formal language theory for the purpose of analyzing complex or essentially nonnumeric types of patterns. While not conceptually novel (pattern recognition techniques with a flavor of the linguistic approach predate this report by over ten years [31]), this approach has only recently begun to receive serious attention. The surface has been scratched sufficiently to suggest possible practical value of the linguistic approach (see, for example, [32], [33]).

Much of the work in this area to date has been directed toward describing patterns of interest by means of grammars, the basic constructs of linguistics. Relatively little attention has been given to the complimentary aspect of the problem, the syntactic (structural) analysis of linguistic representations of patterns; and the work which <u>has</u> been directed toward analysis has clearly demonstrated how indivisible the two aspects of the problem really are: effective pattern analysis depends critically on effective pattern representation. Of course, this is hardly a surprising development considering past experience in pattern recognition research.

The primary emphasis in Chapters 7, 8, and 9 of this report is on the syntactic analysis of patterns, although as suggested above, it is impossible to ignore the linguistic description of patterns. The new material is necessarily very basic, an attempt to lay a foundation for further work in an area in which much remains to be done before practical operational systems can be realized. The remainder of this chapter consists of an introduction to some of the basic definitions of formal linguistics and an overview of the state-of-the-art in linguistic pattern recognition. Chapter 8 considers some relationships between grammars and automata which suggest the use of automata as relatively simple recognizers for the languages generated by grammars. A probabilistic generalization of these relationships is also studied, and some results concerning the capabilities and limitations of automaton recognizers are developed.

Chapter 9 proposes the application of programmed grammars to linguistic pattern recognition. The programmed grammar is a relatively new linguistic formalism which is at once powerful (in terms of the class of languages generated) and convenient to use. Algorithms are developed for generating and analyzing languages in terms of programmed grammars, and a probabilistic generalization is developed which may be especially suitable for dealing with distorted or noisy pattern data.

7.1 Some Basic Concepts of Formal Linguistics

This section presents some of the definitions and terminology of formal language theory which will be used extensively throughout

the balance of the report. Additional material of this nature will be introduced as needed.

The central idea of formal language theory is the generation and analysis of the strings (or sentences) of languages in terms of grammars, typically phrase-structure grammars.

Definition 7.1: A phrase-structure grammar is a 4-tuple

$$G = (V_N, V_T, P, S)$$

where

V_N is a finite set of nonterminals (or variables),

V_T is a finite set of terminals,

P is a finite set of productions (or rewriting rules),

$S \in V_N$ is the start symbol (or sentence symbol).

The productions have the general form $\gamma \rightarrow \delta$, where γ and δ are strings over $V_N \cup V_T$. The symbol "$\rightarrow$" is read "is rewritten as" or "is replaced by".

If V is any set of symbols, V^+ denotes the set of all strings consisting of symbols in V; V^* denotes the set $V^+ + \{\lambda\}$, where λ is the empty string.

Let γ_i and γ_j be two strings from $V_N \cup V_T$. Then $\gamma_i \Longrightarrow \gamma_j$ denotes that γ_j is derived from γ_i by the application of a single production, whereas $\gamma_i \overset{*}{\Longrightarrow} \gamma_j$ denotes that γ_j is derived from γ_i by the application of a sequence of productions ($\Longrightarrow$ is therefore a special case of $\overset{*}{\Longrightarrow}$).

Definition 7.2: Let G be a phrase-structure grammar. The language generated by G, denoted L(G), is defined to be the set of all terminal strings generated by G; i.e.,

$$L(G) = \{x \mid x \in V_T^* \text{ and } S \overset{*}{\Longrightarrow} x\}.$$

Any string generated by a grammar is called a sentential form (a terminal string is a special case). If γ is a sentential form, $|\gamma|$ denotes the length of γ, i.e., the number of symbols in γ.

Definition 7.3: Let G be a phrase-structure grammar, with productions of the form $\gamma \to \delta$:

(a) If no additional restrictions are placed on the productions, then G is an unrestricted (type 0) grammar.

(b) If all of the productions satisfy the additional constraint $|\gamma| \leq |\delta|$, then G is a context-sensitive (type 1) grammar.†

(c) If all of the productions satisfy the constraints

$$\gamma \in V_N \text{ and } \delta \in V_N \cup V_T^+$$

(the left-hand side is a single nonterminal and the right-hand side is a nonempty string), then G is a context-free (type 2) grammar.

(d) If all of the productions have the form

$$A \to aB \text{ or } A \to a$$

where $A, B \in V_N$ and $a \in V_T$, then G is a finite-state (regular or type 3) grammar.

---

†The term "context-sensitive" arises from the fact that typical productions in a context-sensitive grammar may have the form $\gamma_1 A \gamma_2 \to \gamma_1 \gamma_3 \gamma_2$, such a production being applicable to $A \in V_N$ only when it appears in the context $\gamma_1 A \gamma_2$ ($\gamma_1, \gamma_2 \in \{V_N \cup V_T^*\}$).

A language which can be generated by a finite-state grammar is called a finite-state language; context-free, context sensitive, and unrestricted languages are similarly defined. The types of languages form a hierarchy in the sense that: If L is a finite-state language, it is context-free; if L is context-free, it is context-sensitive; if L is context-sensitive, it is unrestricted. In other words, the class of type i grammars generates a more general class of languages than does the class of type j grammars for $i < j$ (the type i grammars are said to be "more powerful" than the type j grammars).

Throughout this and the following chapters, the term "grammar" will mean phrase-structure grammar unless otherwise specifically qualified. "Conventional grammar" will refer to grammars such as those discussed in this section and Chapter 8 as differentiated from the programmed grammars of Chapter 9 (programmed grammars are also phrase-structure grammars, however).

A structural description of the strings of a language in terms of a grammar is called a syntax of the language; the process of analyzing a string in terms of a grammar is called syntactic analysis or parsing.

In formal language theory, the only relation between the symbols in a sentential form is concatenation, the juxtaposition of adjacent symbols. It will be seen in the next section that an important point involved in adapting the techniques of formal language theory to pattern analysis is the generalization of this simple notion to include other relationships and mechanization of the resulting

considerably more general formalism in a systematic way as has been done for formal "string" languages and grammars.

7.2 Literature Review

This section contains a brief look at past developments in the linguistic approach to pattern recognition and an evaluation of the state-of-the-art. A more extensive treatment may be found in a review paper by Fu and Swain [34].

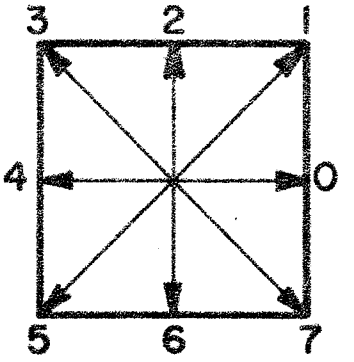
7.2.1 Linguistic Pattern Primitives

The first step in formulating a linguistic model for pattern analysis is the determination of a set of primitives in terms of which the data of interest may be described. This is largely influenced by the nature of the data, the specific application in question, and the technology available for implementing the analysis. The primitives must provide an adequate description of the data, i.e., the primitive descriptions of patterns from different classes must be distinguishable by whatever method is to be applied to analyze the descriptions. In addition to this obvious requirement, the primitive descriptions must be readily obtainable from the raw data and must be compact enough so as not to overtax the memory and processing capacities of the analysis system. In many instances these are conflicting requirements, and it may be necessary to find an acceptable trade-off among them.

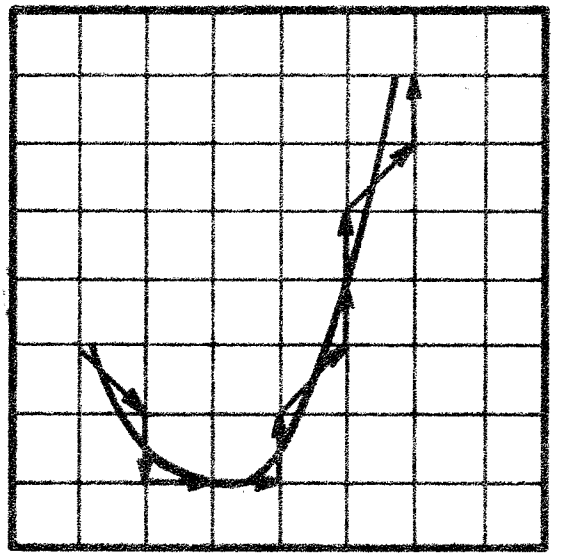
Most of the approaches to the primitive selection problem may be grouped roughly as follows: general methods emphasizing

boundaries, general methods emphasizing regions, and special methods which take advantage of the peculiarities of specific applications. A set of primitives commonly used to describe two-dimensional boundaries is the chain code due to Freeman [35]. Under this scheme, a rectangular grid is overlaid on the two-dimensional pattern and straight line segments are used to connect the grid points falling closest to the pattern (Figure 7.1). Each line segment is assigned an octal digit according to its slope. The pattern is thus represented by a (possibly multiply connected) chain or chains of octal digits. This coding scheme has some useful properties. For example, patterns coded in this way can be rotated through multiples of 45 degrees simply by adding an octal digit (modulo 8) to every digit in the chain (although only rotations by multiples of 90 degrees can be accomplished without some distortion of the pattern). Other simple manipulations such as enlargement, measurement of curve length, and determination of pattern self-intersections are easily carried out. Any desired degree of resolution can be obtained by adjusting the fineness of the grid imposed on the patterns. This method is not limited to closed boundaries; it can be used for coding arbitrary two-dimensional figures composed of straight or curved lines and line segments.

A notable application of Freeman's chain code is the work by Knoke and Wiley [36], which is an attempt to apply linguistic pattern analysis to hand-printed letters. After preprocessing of a CRT-input pattern to obtain a Freeman-code description of the pattern, they use a "transformational grammar" to smooth the pattern. The result of



Octal Primitives

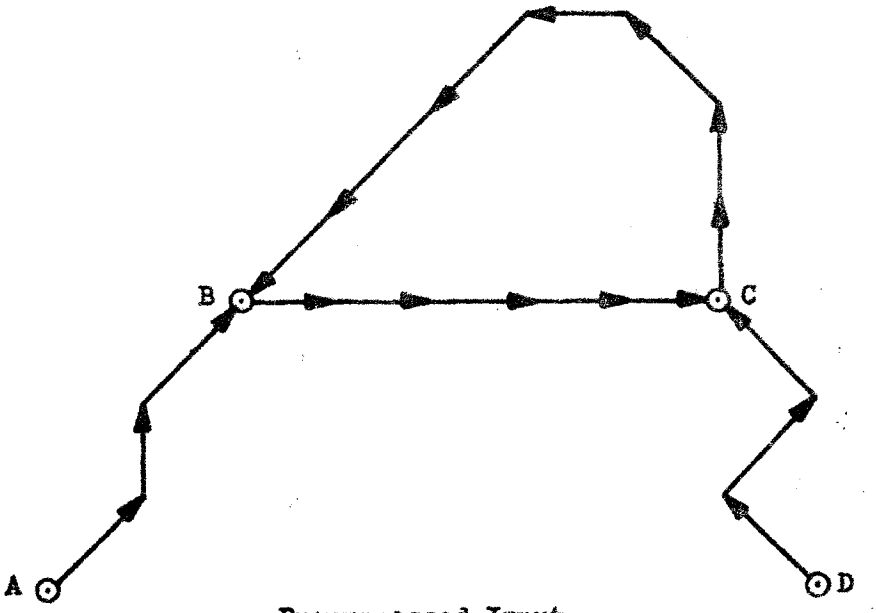


Coded Curve = 7600212212

Figure 7.1 Freeman's Chain Code

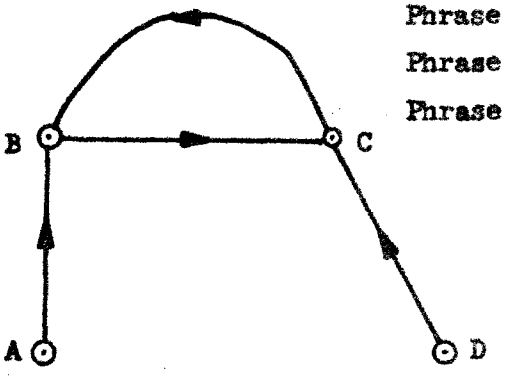
this step is a description of the pattern in terms of a list of vertex interconnections and the shapes of the corresponding connecting arcs, constituting a sentence in the pattern language (Figure 7.2). The sentence is then checked against a dictionary of prototypes and "recognized" if a match is found. Perhaps the most serious disadvantage of their approach, at least from a practical point of view, is the use of a dictionary to store analysis prototypes for matching with unknowns. This sort of approach ("template matching") normally exhibits poor generalization capability (the ability to recognize patterns which vary slightly from the prototypes) and/or requires impractical amounts of memory to store an adequate dictionary. Although the system uses a grammar for reducing the raw pattern to a more refined syntactic description, the syntactics play no explicit role in the actual classification procedure. It will be seen later that the pattern syntactics can be very useful in the latter respect.

A set of primitives for encoding geometric patterns in terms of regions has been proposed by Pavlidis [37]. In this case, the basic primitives are halfplanes in two-dimensional pattern space (this could be generalized to halfspaces of multidimensional pattern space). Any polygon may be interpreted as the union of a finite number of convex polygons, and each convex polygon can in turn be interpreted as the intersection of a finite number of halfplanes. By defining a suitable ordering on the convex polygons composing the arbitrary polygon, it is possible to determine a unique minimal set of maximal (in an appropriate sense) polygons, called primary subsets, the union of which is the



Preprocessed Input

	Arc-Shape	Relative Direction	Initial Vertex	Terminal Vertex
Phrase 1	LIN	0	A	B
Phrase 2	LIN	2	B	C
Phrase 3	CURL	6	C	B
Phrase 4	LIN	7	D	C



"Parsed" Input

Figure 7.2 Parse of a Chain-Encoded Pattern [36]

given polygon. Pavlidis' approach provides a formalism for describing the syntax of polygonal figures and more general figures which can be approximated reasonably well by polygonal figures. However, his analysis procedures require the definition of suitable measures of similarity between polygons. The similarity measures he has considered are quite sensitive to noise in the patterns and some are very difficult to implement practically on a digital camputer.

Rosenfeld and Strong [38] have developed an approach which is applicable when the syntactics involve the relationships of regions but do not explicitly involve their shapes. The pattern primitives are regions bounded by simple closed curves. A hierarchy of syntactic types is based on the containment of regions by other regions. The containment and adjacency relations of regions are expressed in terms of a special graph which can be syntactically analyzed by a new graph-grammatical formalism called a web grammar (discussed later in this section). The generality of this approach is attractive; its practical applicability remains to be demonstrated.

Some interesting but much less general examples of selections of pattern primitives may be found in [32], [39], and [40].

One aspect which most of the approaches to linguistic pattern recognition share is a concern with the reduction of the pattern to be analyzed to some sort of string representation. This is partly due to the relative ease with which string representations can be handled, but it may also be attributed to a desire to take advantage of the existing results in formal language theory, since to use

these results it is necessary either to develop effective methods to accomplish the reduction of patterns to strings of concatenated primitives or to extend the existing theory to include syntactic relations more general than concatenation. The former has to date been the most common approach. One effort to formulate generalized syntactic relations has recently been undertaken by Anderson [41] for the syntactic analysis of handwritten mathematical expressions. Anderson has proposed an entity called a syntactic unit as the multidimensional analog of the one-dimensional syntactic variable (he defines both terminal and non-terminal syntactic units). A syntactic unit is a subscripted variable, say A_p, where A is the name of the unit and p is an ordered set of coordinates specifying the location of the subpattern named A in the pattern space. A pattern described as a list of syntactic units differs in a very fundamental way from the conventional string: Since each syntactic unit contains its own location information, the position of the unit in the list no longer has any particular significance. This gives considerably more flexibility and power to the corresponding grammars than one might expect.

Narasimhan, one of the pioneers of the linguistic approach to pattern recognition, has developed a technique which seems fundamentally different from those mentioned thus far: the parallel processing of entire digital picture arrays [42] - [45]. Under this picture-processing formalism, entire picture arrays are treated as operands of picture-processing operations which produce a new picture array as a resultant. Pictures can thus serve as syntactic primitives and

higher order syntactic constructs as well. Near the top of the hierarchy, however, the syntactic elements are very much like Anderson's syntactic units, each construct consisting of a name and an attribute list (see especially [43]). Interestingly, Pfaltz and Rosenfeld [46] have noted that many (perhaps most) useful picture transformations which have been defined under parallel processing formalisms involve local operations (i.e., functions which define a value for each element in the new picture in terms of the values of the corresponding element and a small set of its neighbors in the original picture), and they have proven that parallel processing of this sort is computationally equivalent to sequential processing. In other words, every local parallel processing function can be accomplished by local sequential processing, and vice versa. What is perhaps most surprising is that sequential processors can be significantly more efficient in terms of the total number of operations involved (the number of points times the number of operations per point) than parallel processors in performing many useful types of picture processing. In fact, parallel processors are sometimes forced to simulate sequential processing and can only do so very inefficiently. But one can still expect that, in general, parallel processing may yield a significant time advantage over sequential processing in spite of the relative computational efficiency of the latter.

Now, assuming that a satisfactory set of pattern primitives is available for a given application, there remain the problems

of specifying the pattern grammar and devising a procedure for the analysis of the primitive pattern description. (Of course, this is not to imply that the primitive selection and pattern analysis aspects can be separated in practice; but the fact that they cannot will not prohibit a dichotomous survey of the work that has been attempted.) Before dealing with this part of the problem it will be well to recall a bit more terminology commonly associated with methods of syntactic analysis.

A convenient way to exhibit syntactic structure is by means of a parsing tree. The root of the tree, conventionally displayed at the top, is labelled with the "start symbol;" the terminating branches, at the bottom, are labelled with terminal symbols the union of which (concatenation in the case of string languages) is a sentence of the language. The structure leading from root to terminating branches describes how the sentence is derived from the start symbol by successive applications of the productions in the grammar.

Some parsing techniques develop the parse by proceeding downward from the root of the tree, i.e., by attempting to synthesize the pattern through use of the syntactic productions. These are called top-down methods. Other methods work upward from the bottom of the tree, attempting to decompose the pattern by applying the productions in reverse. These are called bottom-up methods. There has been considerable debate among formal linguists as to the relative merits of the two approaches for automatic parsing, but from a practical point of view the best approach almost invariably depends on the particular form of the grammar involved [47].

One can describe a "hierarchy" of linguistic or syntactic analysis procedures (Figure 7.3). Any procedure which is at least incidentally concerned with decomposing a pattern into structural or syntactic components will be called here a linguistic procedure. A procedure which has as its specific goal the production and use of the syntactic description of an entity (sentence, pattern) is called a syntax-directed procedure. If the procedure explicitly utilizes a grammar in the process of obtaining the syntactic description (e.g., the grammar may in some manner direct the analysis), then the procedure is said to be syntax-controlled. (This terminology is intended to be similar to that used with respect to syntax-directed compilers [48].)

7.2.2 Pattern Grammars

The Picture Description Language (PDL) developed by Shaw [33], [49] is a reasonably general and flexible formalism for the description and analysis of picture data (and possibly other forms of multi-dimensional data with syntactic content). The PDL is a string grammar: By means of the PDL formalism, one-, two-, and three-dimensional patterns can be put into string form by requiring that all syntactic elements--pictures, subpictures, and primitives as well--have a head and a tail, concatenation occurring only at the head and tail of the syntactic elements (head-to-tail). All pictures are assumed connected, which is made to hold even for pictures with disjoint elements by the definition of suitable blank primitives. A null-point primitive is also defined which consists only of coincident

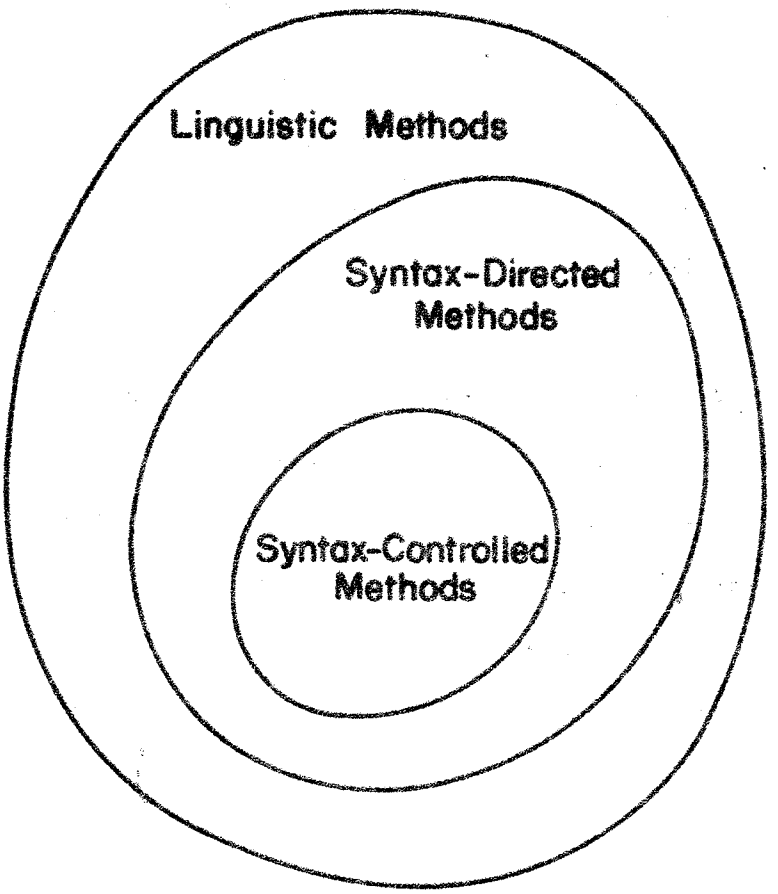


Figure 7.3 A Hierarchy of Linguistic Analysis Techniques

head and tail. Four binary operators together with the null-point primitive are sufficient to describe all possible concatenations of syntactic elements (Figure 7.4). Two unary operators are defined, one of which acts as a "reverser" (head/tail interchange), the other specifying superposition of primitives. The superposition operator in conjunction with the definition of blank primitives and label operators provide the PDL with the ability to describe multiply connected figures. Thirteen context-free productions generate all valid PDL sentences. A PDL grammar for a particular application results when sets of suitable primitives and applicable productions are specified (Figure 7.5). Shaw gives several examples.

Possibly the most severe limitation of the PDL is its restriction to context-free languages. There is a strong feeling in some quarters that context-free languages cannot be expected to provide adequate models for a very general class of complex pattern analysis problems. But this is an open question. The full potential of the PDL deserves to be investigated in this respect. The PDL has some other relatively minor limitations which are discussed by Shaw but do not seem insurmountable. It is significant that the PDL has proved useful in at least one practical application [33] for which a picture analyzer based on the PDL has been implemented. The nature of this implementation is interesting and will be discussed below under Pattern Analysis Mechanisms.

The web grammar [50], [51] offers a natural two-dimensional generalization of string grammars. A web is a directed graph with

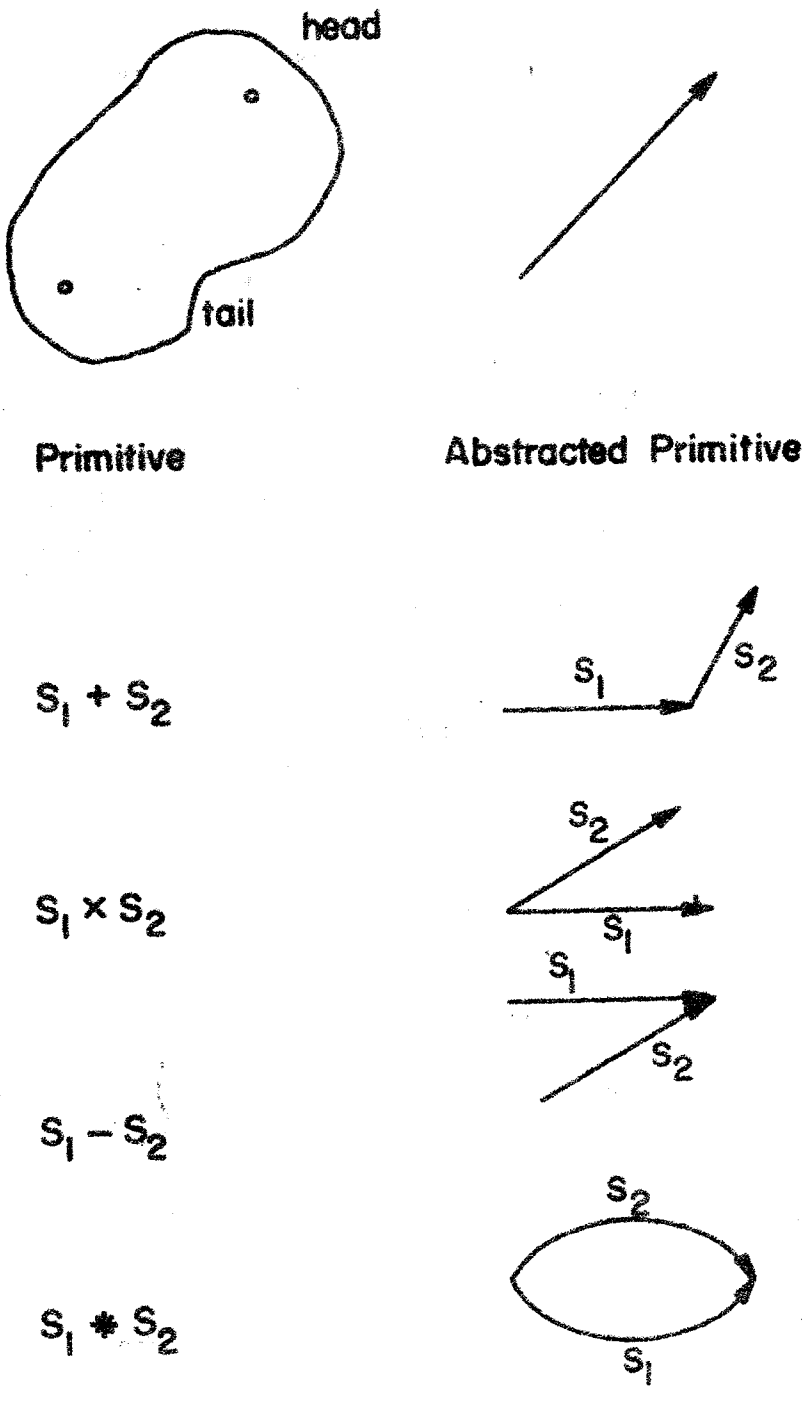


Figure 7.4 A Partial Definition of the PDL [49]

V_T = { dp  , dm  , h  , vm  }

V_N = {Triangle, House, S, A}

P: S → A

S → House

House → ((vm + (h + (~ vm))) * Triangle)

A → (dp + (Triangle + dm))

Triangle → ((dp + dm) * h)

$S \overset{*}{\Longrightarrow}$ ((vm + (h + (~ vm))) * ((dp + dm) * h))

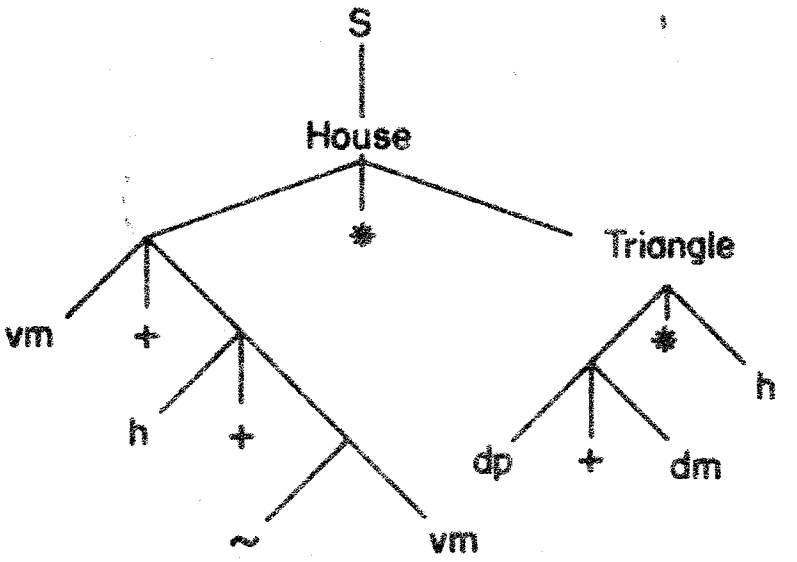


Figure 7.5 A PDL Grammar Generating Houses, A's, and Triangles [49]

nodes bearing labels from a vocabulary of symbols. Since a string is a special case of a graph, a string of symbols is a special case of a web. A web grammar consists of a vocabulary with terminal and non-terminal elements, and a set of rewriting rules (productions)--much like the familiar phrase-structure string grammar. However, the productions of the web grammar are more complicated: Each rule specifies the replacement of a subweb in the original web by another subweb to form the rewritten web. The rule must specify precisely how the new subweb is to be "embedded" in the "host" web. This can be done in any number of ways as long as the embedding rule does not depend on the host web, a requirement which assures that the rule can be applied to any occurrence of the variable (subweb) being rewritten (Figure 7.6). Context-sensitive and context-free web grammars are defined.

Shaw's PDL is a special case of a web grammar, but the web grammar formalism seems to provide a more natural way of representing multiply connected graphs with context-sensitive syntax. Although the practical applicability of web grammars remains to be demonstrated, there would seem to be no reason why this formalism should not be at least as useful as the PDL. Thus, web grammars represent another step toward the generalization of phrase-structure grammatical concepts from strings to more general syntactic entities.

7.2.3 Pattern Analysis Mechanisms

Surprisingly, many of the approaches which have been taken to syntactic pattern analysis are barely syntax-directed, much less

$G = \{V_N, V_T, S, P\}$

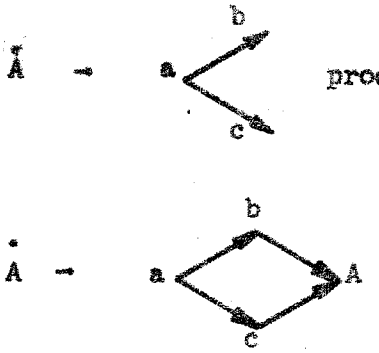
where

$V_N = \{A\}$ nonterminal vocabulary

$V_T = \{a,b,c\}$ terminal vocabulary

$S = \{\dot{A}\}$ initial web

P: productions



where the embedding rule for both productions is

$E = \{(p,a) \mid (p,A)$ is an edge of the host web$\}$

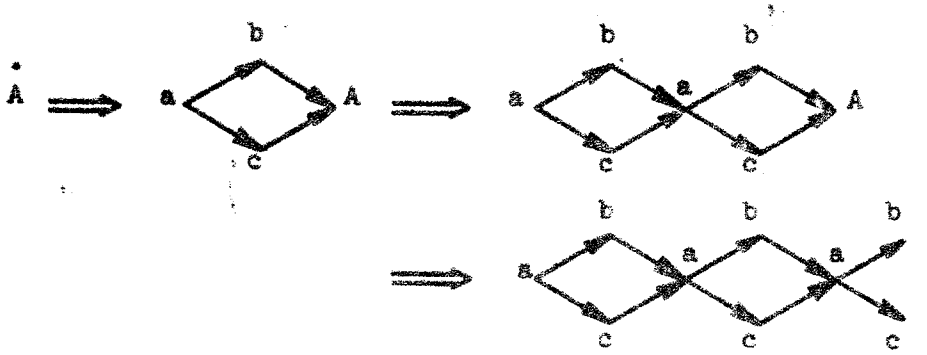


Figure 7.6 A Simple Web Grammar [50]

syntax-controlled, and generally fail to make extensive use for analysis purposes of the considerable information about patterns which is often implicit in pattern grammars. The most promising approaches tend to be syntax-controlled. An example of the success which can be achieved with syntax-controlled pattern recognition is the FIDAC system (Film Input to Digital Automatic Computer) for the detection and classification of chromosome types from photomicrographs [32]. In this case, pattern primitives have been determined (line segments of various shapes) which are at once adequate for the characterization of the patterns and readily detectable by a suitable combination of hardware and computer software. The primitive-encoded patterns amount to strings which can be generated by a fairly simple context-free grammar (Figure 7.7). The analysis procedure parses the input in terms of the grammar. A successful parse automatically accomplishes the classification of a chromosome, because each such parse leads (bottom-up) to a syntactic variable representing a chromosome type.

Anderson [41] uses a syntax-controlled top-down analysis procedure to parse handwritten mathematical expressions (see the earlier discussion of his formalism). His grammars have been carefully formulated to enhance parsing efficiency, which is essential since top-down procedures may be very inefficient, particularly in rejecting "ungrammatical" sentences. He takes advantage of the fact that mathematical expressions tend to be written in a left-to-right manner and also utilizes to a limited extent some of the techniques of

V_T = {a,b,c,d,e}

V_N = {S, T, Bottom, Side, Armpair, Rightpart, Leftpart, Arm}

P: S → Armpair · Armpair
T → Bottom · Armpair
Armpair → Side · Armpair
Armpair → Armpair · Side
Armpair → Arm · Rightpart
Armpair → Leftpart · Arm
Leftpart → Arm · c
Rightpart → c · Arm
Bottom → b · Bottom
Bottom → Bottom · b
Bottom → c
Side → b · Side
Side → Side · b
Side → b · d
Arm → b · Arm
Arm → Arm · b
Arm → a

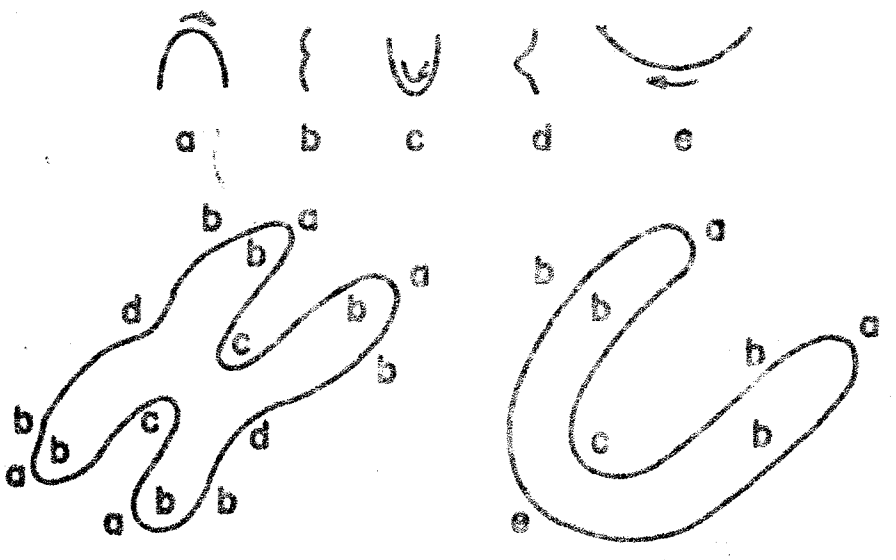


Figure 7.7 The Grammar for FIDAC [32]

precedence analysis [52]. The idea is to arrange the grammar and the analysis procedure so that it becomes obvious as quickly as possible that the analysis is proceeding down a fruitless branch of the parsing tree. Although no precise timing information is provided, Anderson indicates that his analysis procedure is fast enough for the application even though it has not been rigorously optimized.

To implement a pattern analysis system based on the PDL, Shaw [33] has developed a syntax-controlled, top-down, goal-oriented picture parsing algorithm. The algorithm is described as goal-oriented because at each stage of the analysis it tries to attain a a specific goal, namely, to determine whether a specific syntactic construct--the right-hand side of the production being applied--is contained in the raw input pattern. A successful generation of a PDL description of the input pattern indicates that the pattern belongs to the pattern class of interest. The analysis may also yield additional useful information about the pattern depending on any semantic analysis carried out in conjunction with the syntactic analysis.

As pointed out by Shaw, a goal-oriented procedure of this nature has some important advantages over other approaches. The logic of the syntactic analysis is straightforward, consisting primarily of stepping through the pattern grammar, with backtracking as necessary in case false goals are generated at higher levels in the analysis. Also, it is helpful to use a goal-oriented procedure in dealing directly with pattern data which has not been preprocessed

because the syntax specifically prescribes which primitives must be located in the pattern and what relationship these primitives must have to the rest of the pattern. This is, in effect, one way of utilizing context to aid in the recognition process which, as a result, may not need to be as precise or as exhaustive as methods which must search out arbitrary primitives (the job of some pre-processors, for instance, is to compile lists of possible primitives in the pattern without reference to the syntax). Utilizing the syntax may also, in similar fashion, reduce the effects of noise in the data.

Shaw has applied a PDL grammar to the analysis of photographs of atomic particle activities in spark chambers. This practical application has provided one of the most convincing demonstrations available that syntactic pattern recognition is a feasible approach to problems involving complexly structured data.

7.2.4 Learning Pattern Grammars by Grammatical Inference

The researcher's intuition and familiarity with the data are relied on heavily at present in formulating grammars for specific applications. The grammar synthesis procedure is now largely trial-and-error, sometimes aided by user-interactive computer graphics [36], [53]. A method is needed for automatically inferring grammars from representative samples of the languages in question, much as the training procedures of "conventional" pattern recognition technology are used to synthesize pattern classifiers from training samples of known classification. This is apparently a very difficult problem

which has only recently begun to receive attention in the literature [54] - [59].

7.2.5 The Roles of Probability in Syntactic Pattern Analysis

The syntactic pattern analyzers found in the literature are most often well shielded from the "real world" by extensive preprocessing of the raw patterns (smoothing, gap-filling, thinning, etc.) intended to extract pure patterns and eliminate noise. For example, see [32], [36], [42], [53]. The preprocessing techniques commonly used make little or no use of the pattern syntax even though, as noted above, the syntax can be of great assistance in distinguishing noise and detecting distortions of pure patterns. Unfortunately, however, it is precisely distortion and noise which make quite difficult the description of "real life" patterns by means of familiar deterministic models. Recalling that probabilistic approaches have often yielded some success in dealing with distortion and noise, one is led to consider the formulation of a probabilistic model for syntactic analysis.

In attempting to build a universal theoretical foundation for "grammatical analysis of patterns," Grenander [60] has proposed a model called a probabilistic deformation grammar. The model is somewhat vaguely specified in order to maintain generality, but it essentially involves a set of deformation transformations applied to the syntactic variables. A probability measure is defined over the set of deformation transformations. This model appears to be applicable only to the variety of syntactic analysis procedures which do

not depend on a parse of the patterns but only on the description of the patterns in terms of primitives and deformations. These procedures sometimes turn out to be variations on the familiar minimum distance recognition rule.

In another work, Grenander [61] has suggested the possibility of defining a probability distribution over the productions of a context-free pattern grammar. This in turn impresses a probability distribution on the set of parsing trees which may be generated by the grammar and thence on the language generated by the grammar. The model turns out to be a multitype branching process which has been studied in some detail [62]. Grenander has investigated the constraints which must be satisfied by the production probability distribution in order to guarantee a valid probability measure over the set of terminating parsing trees generated by context-free grammars (see also [63]). This model comes much closer in spirit to the concept of syntax-controlled pattern analysis than does the probabilistic deformation grammar, although only empirical studies can be expected to demonstrate which model is truly most appropriate for a given problem. In addition to the work by Grenander and Booth, Kherts [64] has defined and studied briefly the entropy of stochastic context-free grammars; and Fu and Li [65] have described the realization of a stochastic finite-state automaton which recognizes a stochastic finite-state language and have shown how such an automaton can be synthesized if a grammar generating the language is known. Probabilistic grammars will receive further attention in Chapters 8 and 9 of this report.

To summarize this overview: Linguistic methods offer a promising approach to dealing with pattern data which cannot be conveniently described numerically or are otherwise so complex as to defy analysis by conventional techniques, provided that a structural description of such data can be found which contains the information essential to the analysis task. Research in linguistic pattern recognition appears to have demonstrated the feasibility of this approach for some non-trivial applications. However some formidable hurdles remain to be cleared before linguistic pattern recognition can be widely applied. Some of the problem areas are outlined below.

1. Grammar synthesis based on samples of pattern data. Intuition and familiarity with the data have been relied on heavily thus far for deriving pattern grammars. Procedures must be developed for systematically abstracting from training samples "good" sets of pattern primitives, syntactic variables and relationships, and rewriting rules. This can only be done effectively if suitable measures of the "goodness" of pattern grammars can be defined, however, and such measures have yet to be determined.

2. Development of efficient analysis mechanisms for pattern languages. One possible approach is the recognition of patterns by automaton-like devices, a convenient sequential processing procedure. Of course, to analyze patterns characterized by generalized grammatical formalisms such as web grammars, corresponding generalized forms of automata may need to be developed.

An alternative approach is the continued development of special pattern languages and grammars which lend themselves to particularly efficient analysis. The development of such languages and grammars which are in addition powerful enough to characterize a range of interesting types of patterns is a substantial challenge.

3. Detailed formulation of a probabilistic syntactic pattern analysis model for dealing with "real life" patterns corrupted by distortion and random noise.

The material in the next two chapters is intended as a contribution toward the second and third of these problem areas.

CHAPTER 8

AUTOMATA AS RECOGNIZERS OF PATTERN LANGUAGES

Problem: Given a phrase-structure grammar (e.g., a pattern grammar) and a string of symbols from the terminal vocabulary of the grammar, it is desired to determine whether the string is a member of the language generated by the grammar.

As discussed in the preceding chapter, one approach to this problem is to parse the string in terms of the grammar, a process which yields not only an "acceptance" or "rejection" of the string but, in the former case, a syntactic analysis of the string in terms of the grammar. An alternative approach which can be used if an explicit syntactic analysis is unnecessary and which may in general be simpler to implement is to design an automaton which "accepts" only members of the language generated by the grammar and "rejects" all others. In theory such automata can always be designed for context-sensitive phrase-structure grammars [66].

The purpose of this chapter is to explicitly catalog some known results concerning the relationships between grammars and automata and to expand somewhat on these results, particularly with respect to the recently formulated probabilistic (or stochastic) grammars. Some pertinent observations are made concerning the practical application of automata to recognizing the languages

generated by probabilistic as well as nonprobabilistic phrase-structure grammars, and in particular, ways in which automata theory could be applied to pattern recognition under the assumption that patterns of interest may be represented as the elements of languages generated by pattern grammars.

8.1 Finite-State Grammars and Finite Automata

The simplest classes of grammars and automata are finite-state. Finite-state grammars and languages were defined in Section 7.1.

Definition 8.1: A finite automaton is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$ where

Q is a finite set of internal states,

Σ is a finite input alphabet,

$\delta: Q \times \Sigma \to 2^Q$ is the state transition function,

$q_0 \in Q$ is the initial state,

$F \subseteq Q$ is the set of "final states."

If $\delta(q, \sigma)$ is single-valued for every $q \in Q$ and $\sigma \in \Sigma$, then M is deterministic; otherwise M is nondeterministic.

Definition 8.2: A finite automaton M accepts a string x if $\delta(q_0, x) \in F$; i.e., if applying x as input to M in its initial state causes M to end up in one of its final states. The set of all strings accepted by M is denoted by T(M):

$$T(M) = \{x \mid \delta(q_0, x) \in F\}.$$

The relation between grammars and automata will now be stated.

Definition 8.3: The language L(G) generated by a grammar G is said to be recognized by an automaton M if L(G) = T(M); i.e., if M accepts any $x \in L(G)$ but does not accept any $x \notin L(G)$.

The following is a well known result [66].

Theorem 8.1: For every finite-state grammar G there exists a finite automaton M which recognizes L(G).

In particular, if $G = (V_N, V_T, P, S)$ then take $M = (Q, \Sigma, \delta, q_o, F)$ such that

1. $Q = V_N \cup \{T, R\}$; i.e., there is a distinct state of M corresponding to each nonterminal of G, plus an "accept state" T and a "reject state" R;

2. $\Sigma = V_T$; i.e., the input alphabet of M is identical to the set of terminals of G;

3. $q_o = S$; i.e., M starts in the state corresponding to the start symbol of G;

4. $F = \{T\}$; i.e., M accepts all and only strings which cause it to end up in the accept state;*

5. δ, the state transition function, is specified as follows:

   i) $C \in \delta(B, a)$ for all $a \in V_T$ and $B, C \in V_N$ such that $B \rightarrow aC$ is a production of G;

   ii) $T \in \delta(B, a)$ for all $a \in V_T$ and $B \in V_N$ such that $B \rightarrow a$ is a production of G;

*It is convenient to assume that $\lambda \notin L(G)$, but the development presented here can easily be modified to allow $\lambda \in L(G)$.

iii) $R \in \delta(B, a)$ for all $a \in V_T$ and $B \in V_N$ such that neither i nor ii apply;

iv) $\delta(R, a) = \{R\}$ and $\delta(T, a) = \{R\}$ for all $a \in V_T$.

Formulations elsewhere in the literature often omit the reject state R in which case it is inferred that if a state-input combination occurs for which δ is not defined, then the automaton "hangs up" (halts without accepting) and by definition the input string is rejected. For practical purposes, it is convenient to have the automaton pass to the reject state which is a "trap state" by iv.

In general the recognizer described by the above formulation is nondeterministic, which from a practical point of view is undesirable because of the implementation problems which result if the automaton is actually to be constructed. However, the following theorem and subsequent remark obviate this difficulty [66].

Theorem 8.2: If L is a language recognized by a nondeterministic finite automaton, then there exists a deterministic finite automaton which recognizes L.

Furthermore, there is a straightforward procedure for specifying the deterministic recognizer, which may be expected to consist of more states than the nondeterministic version (see [66], pp. 31-32).

Thus it is a fairly simple matter to construct or simulate a recognizer for the language generated by any finite-state grammar. Such a simulation could take one of two forms. The actual sequence of state transitions resulting from any input sequence could be explicitly simulated; or an alternative approach could be used based

on the iterated multiplication of the input-conditioned transition matrices,* which are defined as follows: Let M be an n-state recognizer. The input-conditioned transition matrices for M are given by

$$\{M(\sigma) = [m_{ij}(\sigma)] \mid \sigma \in \Sigma;\ i, j = 1, 2, \ldots, n\}$$

where

$m_{ij}(\sigma) = 1$ if input symbol σ can cause a transition from state i to state j;

$= 0$ otherwise

Higher order transition matrices are defined in the usual way; i.e., for $\sigma \in \Sigma$ and $x \in \Sigma^+$

$$M(\sigma x) = M(\sigma)M(x).$$

The simulation then proceeds in the following manner. Let v_o be an n-component row vector with all components 0 except the component corresponding to the initial state of M, which is 1. Let v_F be an n-component column vector with all components 0 except the component corresponding to the accept state, which is 1 (the only final state is T). Then for any string $x \in \Sigma^+$,

$$v_o M(x) v_F = 0 \quad \text{iff } x \notin T(M) = L(G),$$
$$\geq 1 \quad \text{iff } x \in T(M) = L(G).$$

In fact, the product equals the number of distinct transition sequences which may be caused by the input string or, equivalently, the number of ways in which the string may be generated by the grammar. An advantage of this simulation approach is that it provides a direct

*As distinguished from the transition matrices for finite automata as usually defined; see [67].

and relatively simple simulation of a nondeterministic recognizer, without any need to find the corresponding deterministic version.

Example 8.1: The finite-state language $L_1 = \{00, 11\}^+$ is generated by the finite-state grammar G_1:

$$G_1 = (V_N, V_T, P, S)$$

where

$$V_N = \{A_1, A_2, A_3\}$$

$$V_T = \{0, 1\}$$

$$S = A_1$$

and P consists of

$$A_1 \to 0A_2 \qquad A_2 \to 0$$

$$A_1 \to 1A_3 \qquad A_3 \to 1A_1$$

$$A_2 \to 0A_1 \qquad A_3 \to 1$$

An automaton which recognizes $L_1 = L(G_1)$ is given by

$$M_1 = (Q, \Sigma, \delta, A_1, \{T\})$$

where

$$Q = \{A_1, A_2, A_3, T, R\}$$

$$\Sigma = \{0,1\}$$

and δ is defined by

$$\delta(A_1, 0) = \{A_2\} \qquad \delta(A_3, 0) = \{R\}$$

$$\delta(A_1, 1) = \{A_3\} \qquad \delta(A_3, 1) = \{A_1, T\}$$

$$\left.\begin{array}{ll} \delta(A_2, 0) = \{A_1, T\} & \delta(T, a) = \{R\} \\ \delta(A_2, 1) = \{R\} & \delta(R, a) = \{R\} \end{array}\right\} \text{ any } a \in \Sigma.$$

The input-conditioned transition matrices are given by

$$M(0) = \begin{array}{c} \\ A_1 \\ A_2 \\ A_3 \\ T \\ R \end{array} \begin{array}{c} \begin{array}{ccccc} A_1 & A_2 & A_3 & T & R \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

$$M(1) = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

In practice, the input-conditioned transition matrices may be replaced by the submatrices which have the "reject" row and column deleted, since no input can reach the accept state by any state sequence which includes the reject state.

8.2 Stochastic Finite-State Grammars and Associated Recognizers

Considerable effort is currently being directed toward the formulation of probabilistic grammatical models [60], [61], [63], [65], [68] - [70].

Definition 8.4: A stochastic finite-state grammar (SFSG) G_p is a 5-tuple

$$G_p = (V_N, V_T, P, D, S)$$

where V_N, V_T, P, and S are as given in Definitions 7.1 and 7.2 and D is a rule for assigning a probability measure over the productions.

The detailed specification of D will be of concern in the remainder of this section.

Definition 8.5: Let $x = a_1 a_2 \cdots a_n \in V_T^+$ be a string generated by a SFSG G_p using the sequence of productions $r_1, r_2, \ldots, r_n$. The generation process can be represented as

$$S \xrightarrow{r_1} a_1 A_1 \xrightarrow{r_2} a_1 a_2 A_2 \xrightarrow{r_3} \cdots \xrightarrow{r_n} a_1 a_2 \cdots a_n.$$

The probability associated with the generation is defined to be the product of conditional probabilities

$$p(r_1)p(r_2 \mid r_1) \cdots p(r_n \mid r_1, r_2, \ldots, r_{n-1}).$$

If the string $x \in L(G_p)$ can be generated by k distinct sequences of productions (the grammar is ambiguous if k is greater than 1), then the probability associated with string x is defined as

$$g(x) = \sum_{1}^{k} p(r_1)p(r_2 \mid r_1) \cdots p(r_n \mid r_1, r_2, \ldots, r_{n-1})$$

where the sum is taken over the distinct generations of x.

Definition 8.6: D is a consistent production probability assignment ("D is consistent") provided

$$\sum_{x \in L(G_p)} g(x) = 1.$$

Necessary and sufficient conditions for D to be consistent in the finite-state case are given in [61], [63] and need not be

detailed here. All stochastic grammars considered herein will be assumed to have consistent production probability assignments.

As the notation above indicates, the production probability assignment may depend on the specific sequence of productions used. In the simplest case the assignment is independent of the productions previously applied; that is, it depends on the nonterminal to be rewritten but not on how that nonterminal was obtained. This situation will be denoted by

$$p(r_i \mid r_1, r_2, \cdots, r_{i-1}) = p(r_i)$$

which, for the finite-state case, can be written as

$$p(r_i \mid r_1, r_2, \cdots, r_{i-1}) = p(a_i A_i \mid A_{i-1})$$

(taking $S = A_0$).

Definition 8.7 [63]: D is an unrestricted production probability assignment provided

$$p(r_i \mid r_1, r_2, \cdots, r_{i-1}) = p(r_i)$$

for all production sequences.

Notice that stochastic grammars are no more powerful in terms of the languages generated than their nonprobabilistic counterparts. Stochastic finite-state grammars, for example, still generate only finite-state languages (regular sets). The effect of the added probabilistic machinery is merely to impress a probability distribution on the sentences of the languages generated. Therefore the considerable body of results which has been obtained for nonprobabilistic grammars and languages (e.g., decidability results) is no

less valid for the stochastic case.

Example 8.2: $G_{pl} = (V_N, V_T, P, D, S)$ where $V_N = \{S, A, B, C\}$, $V_T = \{0, 1\}$, and P and D are as shown below.

P	D
$S \to 0A$	$p(0A \mid S) = \alpha \quad (0 < \alpha < 1)$
$S \to 1B$	$p(1B \mid S) = 1 - \alpha$
$A \to 1A$	$p(1A \mid A) = \beta \quad (0 < \beta <$)
$A \to 1C$	$p(1C \mid A) = 1 - \beta$
$B \to 0C$	$p(0C \mid B) = 1$
$C \to 0$	$p(0 \mid C) = 1$

The language generated is

$$L(G_{pl}) = \{100 \cup 01(1)^*0\}$$

$$= \{100, 010, 0110, 01110, \ldots\},$$

and the impressed distribution is given by

$$g(100) = 1 - \alpha$$

$$g(01^k0) = \alpha\,\beta^{k-1}(1 - \beta), \quad k = 1, 2, \ldots.$$

D is an unrestricted production probability assignment; it is also consistent since

$$\sum_{x \in L(G_{pl})} g(x) = (1 - \alpha) + \alpha(1 - \beta) \sum_{k=1}^{\infty} \beta^{k-1}$$

$$= 1 - \alpha + \alpha(1 - \beta) \cdot \frac{1}{1 - \beta}$$

$$= 1$$

Definition 8.8: A stochastic finite automaton (SFA) M_p is a 5-tuple

$$M_p = (Q, \Sigma, \Delta, q_o, F)$$

where Q, Σ, q_o, and F are as in Definition 8.1 and Δ is a stochastic state transition function.

The state transition function Δ can be defined by a set of stochastic matrices $\{\Delta_\sigma \mid \sigma \in \Sigma\}$ which specify the state transitions as a stochastic function of the input symbols. Further, letting $x = \sigma_1\sigma_2\cdots\sigma_k$ be a sequence of length k from Σ^*, Δ can be extended to a function over input sequences according to

$$\Delta(\lambda) = I = \Delta_\lambda \qquad \text{(identity matrix)}$$

$$\Delta(x) = \Delta_{\sigma_1}\Delta_{\sigma_2}\cdots\Delta_{\sigma_k}.$$

Assuming Q contains n states, let v_o be an n-component row vector with all components 0 except the component corresponding to the initial state q_o, which is 1. Then the vector v given by

$$v = v_o\,\Delta(x), \qquad x \in \Sigma^*$$

is an n-component stochastic row vector, the i^{th} component of which is the probability that the input sequence x will cause M_p to pass from the initial state to the i^{th} state. Let v_F be an n-component column vector with all components 0 except the components corresponding to the final states, which are 1. Then p(x), defined by

$$p(x) = v_o\Delta(x)v_F, \qquad x \in \Sigma^*$$

is the probability that the input sequence x will cause M_p, starting

from its initial state, to end up in one of its final states (i.e., a state $q \in F$).

The correspondence between stochastic grammars and stochastic automata depends critically on the definition of acceptance of a string or recognition of a language by automata in a stochastic sense. Let λ be a real-valued parameter, $0 \leq \lambda < 1$, called a cut-point (there should be no confusion between the notations for cut-point and the null string).

Definition 8.9 [Ya]. A sequence x is accepted with cut-point λ by a SFA M_p provided

$$p(x) = v_0 A(x) v_F > \lambda.$$

The set of all strings accepted by M_p with cut-point λ is denoted by $T(M_p, \lambda)$; i.e., $T(M_p, \lambda) = \{x \mid p(x) > \lambda\}$.

Definition 8.10: A language (or set of sequences) L is recognized with cut-point λ by a SFA M_p provided $L = T(M_p, \lambda)$; i.e., provided

$$L = \{x \mid p(x) > \lambda\}$$

$T(M_p, 0)$ is always a finite-state language. The remainder of this section considers only this case. Nonzero cut-points will be discussed in the next section.

The following definition will be useful.

Definition 8.11: Let L be an arbitrary subset of Σ^*, where Σ is the input alphabet of some SFA M_p, and let $g(x)$ be an arbitrary function over L satisfying $0 \leq g(x) \leq 1$. Then M_p is said to compute the function $g(x)$ if $p(x) = v_0 A(x) v_F = g(x)$ for all $x \in L$.

The function g(x) will generally be taken to be the probability of generation of a string by a grammar.

Theorem 8.3: For every stochastic finite-state grammar G_p with an unrestricted and consistent production probability assignment, there exits a stochastic finite automaton M_p which

i) recognizes $L(G_p)$ with cut-point $\lambda = 0$, and

ii) computes the probability of generation g(x) for any $x \in L(G_p)$.

Proof: The proof is by construction. Assume $G_p = (V_N, V_T, P, D, S)$ where D is unrestricted and consistent. Take $M_p = (Q, \Sigma, \Delta, q_o, F)$ such that

1. $Q = V_N \cup \{T, R\}$, where T is the "accept" state and R is the "reject" state;†

2. $\Sigma = V_T$;

3. $q_o = S$;

4. $F = \{T\}$;††

5. Δ, the stochastic state transition function, is given by the set of transition matrices

$$\{\Delta_a = [\delta_{ij}(a)] \mid a \in V_T\}$$

where

$\delta_{ij}(a) = p(aA_j \mid A_i)$ if $A_i \rightarrow aA_j$ is a rule of P with associated probability $p(aA_j \mid A_i)$;

---

†Assume that V_N contains n nonterminals and let the n + 2 states of Q be ordered such that the first n states correspond to the elements of V_N, T is state n + 1 and R is state n + 2.

††It is again assumed for convenience that $\lambda \notin L(G_p)$.

- $\delta_{iT}(a) = p(a \mid A_i)$ if $A_i \rightarrow a$ is a rule of P with associated probability, $p(a \mid A_i)$;
- $\delta_{iR}(a) = 1 - \sum_{j=1}^{n+1} \delta_{ij}(a)$;
- $\delta_{Tj}(a) = \delta_{Rj}(a) = 0, \quad j = 1,2,\ldots T$, all $a \in V_T$

  $\delta_{RR}(a) = \delta_{RR}(a) = 1, \quad$ all $a \in V_T$.

Notice that except for the probabilities assigned to the allowable state transitions, the construction is exactly the same as for the nonprobabilistic case (given in the previous section). Thus i follows immediately from the fact that the set of input sequences x for which $p(x) > 0$ is, as in the nonprobabilistic case, the language generated by the grammar. To prove ii, it must be shown that for all $x \in T(M_p, 0) = L(G_p)$,

$$v_0 \Delta(x) v_F = g(x).$$

In this case v_0 has exactly one nonzero component, namely the one corresponding to the state representing the start symbol S; and v_F has exactly one nonzero component, namely the one corresponding to the accept state T. Therefore, $v_0 \Delta(x) v_F$ is just the element in row S and column T of $\Delta(x)$. The desired result then follows from 5 of the construction and the definition of matrix multiplication.

Example 8.3: For the grammar G_{pl} given in Example 8.2, the SFA which recognizes $L(G_{pl})$ and computes the probability of any $x \in L(G_{pl})$ is:

$$M_{pl} = (Q, \Sigma, \Delta, S, \{T\})$$

where

$$Q = \{S, A, B, C, T, R\}$$

$$\Sigma = \{0, 1\}$$

and Δ is given by the transition matrices

$$\Delta_0 = \begin{array}{c} \\ S \\ A \\ B \\ C \\ T \\ R \end{array} \begin{array}{c} \begin{array}{cccccc} S & A & B & C & T & R \end{array} \\ \begin{bmatrix} 0 & \alpha & 0 & 0 & 0 & 1-\alpha \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

$$\Delta_1 = \begin{bmatrix} 0 & 0 & 1-\alpha & 0 & 0 & \alpha \\ 0 & \beta & 0 & 1-\beta & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

By direct calculation,

$$p(100) = v_0 \Delta_1 (\Delta_0)^2 v_F = 1 - \alpha$$

$$p(01^k0) = v_0 \Delta_0 (\Delta_1)^k \Delta_0 v_F = \alpha \beta^{k-1}(1 - \beta)$$

(compare with Example 8.2).

As an application of these results, suppose that the patterns from two pattern classes ω_1 and ω_2, not necessarily disjoint, can be generated by two stochastic finite-state grammars G_{ω_1} and G_{ω_2}. Assume that the grammars are such that the probability of generation of any pattern is equal to the class-conditional probability of observing that pattern; i.e.,

$$g_{\omega_1}(x) = p(x \mid \omega_1) \quad \text{and} \quad g_{\omega_2}(x) = p(x \mid \omega_2).$$

Let M_{ω_1} and M_{ω_2} be the stochastic finite automata which recognize $L_{\omega_1} = L(G_{\omega_1})$ and $L_{\omega_2} = L(G_{\omega_2})$, respectively, with cut-point $\lambda = 0$ and which compute the probabilities of generation $g_{\omega_1}(x)$ and $g_{\omega_2}(x)$. Then since $g_{\omega_i}(x) = v_{oi}A_i(x)v_{Fi} = p(x \mid \omega_i)$, $i = 1, 2$, the usual minimum risk classification rule (see Section 2.1) can be written: Decide

$$x \in \omega_1 \text{ if } P_1 v_{o1} A_1(x) v_{F1} \geq P_2 v_{o2} A_2(x) v_{F2}$$

$$x \in \omega_2 \text{ otherwise,}$$

where P_1 and P_2 are the a priori probabilities of the classes.

8.3 Recognition With Nonzero Cut-Points

As mentioned in the previous section, any language recognized by a SFA with cut-point $\lambda = 0$ is a finite-state language. However, the class of languages recognized with cut-points $0 \leq \lambda < 1$ properly includes the finite-state languages [71]. Since it is desirable to have available recognition devices for the most general class of languages possible, the case of nonzero cut-points is potentially of interest.

A special kind of nonzero cut-point is defined.

Definition 8.12 [71]: A cut-point λ is called an isolated cut-point with respect to the SFA M_p if there exists an $e > 0$ such that

$$|p(x) - \lambda| \geq e \text{ for all } x \in \Sigma^*$$

(Notice that $\lambda = 0$ cannot be an isolated cut-point.)

It is well known [71] that the class of languages recognized by SFA's with isolated cut-points is exactly the class of finite-state languages. It is also known that some finite-state languages which are recognized by SFA's with isolated cut-points can be recognized by SFA's with fewer states than the minimal nonprobabilistic finite automaton which recognizes the language. Unfortunately, no general procedure is known for finding the minimal SFA for a given finite-state language. Also, some languages which are context-free but not finite-state are accepted by SFA's with nonisolated cut-points [69], but no procedure is known for constructing such automata for given context-free languages, nor has the class of context-free languages accepted by SFA's been completely characterized.

Consider the question as to whether an SFA with nonzero cut-point can be applied to recognizing a language generated by a stochastic finite-state grammar. If the automaton is merely required to recognize the language without computing the probability of generation g(x), then the question of existence of such an automaton is trivial since a deterministic automaton (a SFA with only unity or zero transition probabilities) can be found as shown previously which recognizes the

language, and any cut-point $0 \leq \lambda < 1$ is then an isolated cut-point with respect to such an automaton (because $p(x) = 1$ for $x \in L$ and $p(x) = 0$ for $x \notin L$). However, finding the minimal SFA is a much more difficult--and as yet unsolved-- problem.

The remainder of this section precisely characterizes the class of languages with associated probability measures $g(x)$ which can be recognized by SFA's which compute $g(x)$.

Theorem 8.4: Let L be an infinite language (a language containing an infinite number of strings) with an associated probability measure defined over all $x \in L$. There does not exist a SFA with a nonzero cut-point which recognizes L and computes $g(x)$.*

Proof: Assume that a SFA M_p recognizes L with cut-point λ and computes $g(x)$. Let $x_1, x_2, \ldots$ be an ordering of the strings in L such that $g(x_i) \geq g(x_j)$ for $i < j$. Since M_p computes $g(x)$, it must be true that $p(x_i) \geq p(x_j)$ for $i < j$. Also, since $g(x)$ is a probability measure, $g(x)$ and hence $p(x)$ must satisfy

$$\sum_{i=1}^{\infty} g(x_i) = \sum_{i=1}^{\infty} p(x_i) = 1$$

which requires that

$$\lim_{i \to \infty} p(x_i) = 0$$

Thus, for every $e > 0$, there must be an integer $N = N(e)$ such that

$$p(x_i) < e \text{ for all } i > N.$$

*It is assumed that $g(x) > 0$ if and only if $x \in L$.

But since $x_i \in L$, by definition it must be that $p(x_i) > \lambda$ for all i. Thus

$$\lambda < p(x_i) < e, \quad i > N,$$

or $\lambda < e$. This can hold for every positive e only if $\lambda = 0$.

Corrolary 8.4.1: No SFA with isolated cut-point can both recognize an infinite language L and compute an associated probability measure over L.

Since all finite languages are regular sets, the following--a slightly different point of view--also holds.

Corrolary 8.4.2: Let L be a set with associated probability measure g(x). If there exists a SFA M_p which recognizes L with cut-point $\lambda > 0$ and computes g(x), then L is a finite regular set.

Theorem 8.5: Given a stochastic finite-state grammar G_p with a consistent unrestricted production probability assignment, if $L(G_p)$ is a finite language then a SFA M_p with isolated cut-point λ can be constructed based on G_p such that $T(M_p, \lambda) = L(G_p)$ and M_p computes g(x). Let $\bar{x} \in L(G_p)$ be the string for which

$$p(\bar{x}) = \min_{x \in T(M_p,0)} p(x) = \min_{x \in L(G_p)} g(x)$$

($\bar{x}$ need not be unique). Then any fixed λ satisfying $0 < \lambda < p(\bar{x})$ is an isolated cut-point with respect to M_p (recall the assumption that $g(x) = 0$ for $x \notin L(G_p)$).

The results which have been given in this section are quite restrictive from a theoretical point of view. Whether they are too

restrictive to allow formulation of useful automaton recognizer models (e.g., for pattern recognition) remains to be seen.

8.4 "Maximum-Likelihood" Recognition

The results of the previous section suggest that SFA's with non-zero cut-points may have rather limited utility as recognizers for languages generated by stochastic grammars (more will be said about this later). It has been suggested elsewhere [65] that a slightly different mechanism, which will be referred to as maximum-likelihood recognition, might be of use for such purposes. This possibility is explored here.

Let $M_p = (Q, \Sigma, \Delta, q_o, F)$ be a SFA as defined earlier. Recall that the i^{th} component of the row vector

$$v = v_o\Delta(x) = [p(q_1 \mid x), p(q_2 \mid x), \cdots]$$

is the probability that the input sequence $x \in \Sigma^*$ will cause M_p, beginning in its initial state, to end up in state $q_i \in Q$.

Definition 8.13 [65]: The function $p(\bar{q} \mid x)$ defined by

$$p(\bar{q} \mid x) = \max_{q_i \in Q} p(q_i \mid x)$$

is called the maximum-likelihood membership function (MLMF) of x with respect to M_p.

Definition 8.14: A string x is accepted in the maximum-likelihood sense (ML accepted) by a SFA M_p provided $p(\bar{q} \mid x)$ is such that $\bar{q} \in F$. The set of all strings accepted by M_p in the ML sense is denoted by $T_{ML}(M_p)$.

Definition 8.15: A language L is recognized in the maximum-likelihood sense (ML recognized) by a SFA M_p provided $L = T_{ML}(M_p)$; i.e., provided

$$L = \{x \mid \text{for } p(\bar{q} \mid x),\ \bar{q} \in F\}$$

Any such language is called a maximum-likelihood language (ML language).

Some simple properties of ML languages have been studied by Fu and Li [65]. For example, every finite-state language is trivially a ML language; in fact, the class of ML languages properly includes the class of finite-state languages. Thus, as in the case of "cut-point recognition" discussed above, the ML criterion can be used to make simple accept/reject decisions on the strings generated by a stochastic finite-state grammar. The obvious question to ask is whether it can do more.

Definition 8.16: Let L be an arbitrary subset of Σ^*, where Σ is the input alphabet of some SFA M_p, and let $g(x)$ be an arbitrary function over L satisfying $0 \leq g(x) \leq 1$. Then M_p is said to ML compute $g(x)$ if

$$p(\bar{q} \mid x) = g(x) \text{ for all } x \in L.$$

Unfortunately, the results for this type of recognition are found to be no more helpful than those of the previous section. In the following, $g(x)$ may be thought of as the probability of generation of a string x by a stochastic grammar.

Theorem 8.6: Let L be an infinite language with an associated probability measure defined over all $x \in L$. There does not exist a

SFA which both ML recognizes L and ML computes $g(x)$.

Proof: Assume that a SFA M_p ML recognizes L and ML computes $g(x)$. Let $x_1, x_2, \ldots$ be an ordering of the sequences in L such that $g(x_i) \geq g(x_j)$ for $i < j$. Since M_p ML computes $g(x)$, it must be true that $p(\bar{q} \mid x_i) \geq p(\bar{q} \mid x_j)$ for $i < j$. Also, since $g(x)$ is a probability measure, $g(x)$ and hence $p(\bar{q} \mid x)$ must satisfy

$$\sum_{i=1}^{\infty} g(x_i) = \sum_{i=1}^{\infty} p(\bar{q} \mid x_i) = 1$$

which requires that

$$\lim_{i \to \infty} p(\bar{q} \mid x_i) = 0$$

Thus, for every $e > 0$, there must be an integer $N = N(e)$ such that

$$p(\bar{q} \mid x_i) < e \quad \text{for all } i > N.$$

But for an n-state automaton,

$$p(\bar{q} \mid x_i) = \max_j \; p(q_j \mid x_i) \geq \frac{1}{n} .$$

Combining the last two expressions gives

$$\frac{1}{n} \leq p(\bar{q} \mid x_i) < e$$

which cannot be true for finite n and every positive e. Thus the assumption that M_p ML computes $g(x)$ leads to a contradiction.

Corrolary 8.6.1: Let L be a set with associated probability measure $g(x)$. If there exists a SFA M_p which ML recognizes L and ML computes $g(x)$, then

i) L is a finite regular set;

ii) M_p has at least n states, where n is the smallest integer greater than or equal to $[\min_{x \in L} g(x)]^{-1}$.

8.5 Some Remarks on Application

Given the task of recognizing a language generated by a deterministic finite-state grammar, one might very well synthesize a deterministic finite automaton to perform the recognition (in terms of, say, a suitable sequential circuit [72] - [74]). The "accidental" probabilistic behavior of such a device, due to component unreliability, is not of concern here. But presumably one might consider deliberately synthesizing a stochastic automaton either to realize, in the finite-state language case, the potential saving of states, or possibly even to recognize a nonfinite-state language by means of a finite automaton (see Section 8.3). This raises some important practical questions.

In general (by definition) the response of a stochastic finite automaton to any input sequence occurs with a certain measure of uncertainty. Therefore, if a sequence causes the automaton to end up in a state within the final state set, it can only be said that there is some nonzero probability that the sequence is a member of the language to be recognized. Similarly if the automaton ends up in a state outside the final state set, the sequence still may have a nonzero probability of belonging to the language to be recognized. And in either case, the response alone gives no information as to what the associated probability might be. Thus, given such a device to perform recognition, something like the following probabilistic

experiment described in [71] would have to be performed for any sequence to be tested (assuming the recognition criterion is accepted with cut-point, according to Definition 8.9):

Let M_p be the automaton, λ the cut-point. The sequence x to be tested is applied to M_p (started in its initial state) some large number N of times. If N_a is the number of times that M_p ends up in a final state, then N_a/N is an unbiased estimate of the final state probability $p(x)$. If N_a/N turns out to be greater than λ, the input is assumed to be a member of the language; otherwise it is rejected. Of course, this is still a probabilistic decision, made with a level of confidence which depends on the actual value of $|p(x) - \lambda|$ (among other things). Since $p(x)$ is unknown, the confidence level cannot, in general, be determined, although if the cut-point is isolated the confidence level may be bounded from below.

In view of the above observations, it seems that, quite aside from the difficulties inherent in realizing probabilistic systems, recognition by stochastic automata per se is a hopelessly impractical affair. On the other hand, this does not necessarily diminish the utility of the formal model, which provides a fairly elementary computational algorithm for performing recognition of strings of a language by means of, say, a digital computer. So at least from this point of view, the continued investigation of this approach is certainly still merited, with the awareness that the results are more likely to be applied through simulation on a computer than through the actual construction of stochastic automata.

The foregoing discussion has introduced a viewpoint that the author feels is important. If the "simulation assumption" discussed in the previous paragraph is accepted as reasonable, the entire emphasis in research on this subject may be affected. For example, one result is to remove the stigma which has sometimes seemed to be associated with nonisolated cut-points, including $\lambda = 0$ (viz., the smaller the actual value of $|p(x) - \lambda|$, the larger must be the number of applications of a sequence to the automaton to achieve a given level of confidence in the accept/reject decision; which in turn implies, for a fixed confidence level, that as $|p(x) - \lambda|$ becomes infinitesimal the number of applications must become infinite). In this sense, stochastic automaton models, such as the one introduced in the proof of Theorem 8.3 which can recognize a string and compute the probability of generation of the string by an associated grammar may prove quite useful.

The study of additional "acceptance" criteria may also lead to automaton models of greater practical value (the maximum likelihood criterion was one attempt in this direction).

The material in this chapter has been limited to the study of finite-state languages and finite automata. Some theoretical groundwork for a more general class of languages and automata has also begun to receive attention [68], [70].

CHAPTER 9

STOCHASTIC PROGRAMMED GRAMMARS

The programmed grammar [75], [76] is a powerful and convenient formalism for generating a relatively broad class of languages (see Theorem 9.1). This chapter extends the formalism into the probabilistic realm and discusses an algorithm for the analysis of strings in terms of programmed grammars.

Definition 9.1: A programmed grammar is a 5-tuple

$$G_{pg} = (V_N, V_T, P, J, S)$$

where

V_N is a finite set of variables (nonterminals),

V_T is a finite set of terminals $(V_N \cap V_T = \emptyset)$,

P is a finite set of programmed productions,

J is a finite set of production labels,

$S \in V_N$ is the start symbol.

Each production consists of a label $r \in J$, a core production of the phrase-structure type, and a success branch field and a failure branch field each consisting of elements from J. A derivation or generation in G_{pg} proceeds as follows: the first production is applied to the start symbol S; in general, if production r is

applied to the current sentential form γ to rewrite a substring α and if γ contains at least one occurrence of α, then the leftmost α is rewritten by the core of production r and the next production label is selected from the success branch field of r; if the current sentential form does not contain α, then the core of production r cannot be used and the next production label is selected from the failure branch field of r; if the applicable branch field is empty, the derivation halts.

Definition 9.2 [75]: A context-free programmed grammar (CFPG) is a programmed grammar in which the cores have a single symbol (nonterminal) on the left side and a nonnull string on the right. The set of all strings generated by a CFPG G_{pg} is called the CFPG language generated by G_{pg}.

Example 9.1a: The context-free programmed grammar G_{SQ} given below generates the language $L_{SQ} = \{a^n b^n c^n d^n \mid n \geq 1\}$ which could be interpreted as the language of squares of side length n = 1, 2, ...

$$G_{SQ} = (V_N, V_T, P, J, S)$$

where the vocabulary consists of

$$V_N = \{S, A, B, C, D\}$$

$$V_T = \{a, b, c, d\},$$

the label set is

$$J = \{1, 2, 3, 4, 5, 6, 7\},$$

and the production set P consists of 7 rules:

Label	Core	Success Branch	Failure Branch
1	S → aAB	2,3	∅
2	A → aAC	2,3	∅
3	A → D	4	∅
4	C → d	5	6
5	D → bDc	4	∅
6	B → d	7	∅
7	D → bc	∅	∅

The generation of a string in L_{SQ} proceeds as follows. After production 1 is used, the length n of the side of the square is determined by the number of times production 2 is applied before production 3 is applied. B is an end marker and the C's serve as counters to keep track of the length. Once the first side has been completed, the elements of the other sides are generated until the C's and B have been converted to terminals.

The following context-sensitive phrase-structure grammar also generates L_{SQ}.

Example 9.1b: $G'_{SQ} = (V_N, V_T, P, S)$

where

$V_N = \{S, A, B, C, D, E, F, G\}$

$V_T = \{a, b, c, d\}$

and the production set consists of 19 rules:

1. S → aAB
2. A → aAC
3. A → D
4. Dc → cD

5. $Dd \rightarrow dD$
6. $DC \rightarrow EC$
7. $EC \rightarrow Ed$
8. $DB \rightarrow FB$
9. $Ed \rightarrow Gd$
10. $cG \rightarrow Gc$
11. $dG \rightarrow Gd$
12. $aG \rightarrow abcD$
13. $bG \rightarrow bbcD$
14. $dFB \rightarrow dFd$
15. $dFd \rightarrow Fdd$
16. $cF \rightarrow Fc$
17. $bF \rightarrow bbc$
18. $aF \rightarrow ab$
19. $bB \rightarrow bc$

(The productions are numbered for reference only.) After production 1 is used, the length of the side of the square is increased by production 2 until production 3 is used to rewrite A. Again B is an end marker and the C's serve to keep track of the side length. Productions 4-8 effect a left-right scan of the string to see if any C's remain. If so, a d (production 7) and eventually b and c are generated in appropriate positions (production 12 or 13), and the scan is repeated. If no C's remain, the B is encountered during a left-right scan after which the final b, c, and d are generated (productions 13-17 or 18-19), and the generation halts having produced a string in L_{SQ}.

The grammars G_{SQ} and G'_{SQ} generate the strings of L_{SQ} by much the same procedure, yet the programmed grammar seems intuitively much simpler. The main reason for this, of course, is that the scanning mechanism which must be explicitly simulated by the conventional grammar is an inherent part of the programmed grammar. Other contributing factors are: For the programmed grammar the sequence of productions can be explicitly specified; and it is always the leftmost

instance of a nonterminal which is rewritten by the application of a production. Important consequences of these differences include:

1. CFPG's have greater generative power than conventional context-free grammars (see Theorem 9.1).

2. CFPG's are generally much easier to write for generating a given context-sensitive language than is a conventional context-sensitive grammar.

3. Generative and analytical mechanisms based on CFPG's are apparently much easier to synthesize than those based on conventional context-sensitive grammars.

The first of these points is illustrated by the previous example, since there is no context-free grammar of the conventional type which generates L_{SQ}. Rosenkrantz has proven several theorems concerning the generative power of various forms of programmed grammars. The following theorem is of interest here.

Theorem 9.1 [75]: The set of languages generated by context-free programmed grammars properly contains the set of context-free languages and is properly contained within the set of context-sensitive languages.

Experience indicates that the class of CFPG languages includes interesting context-sensitive languages (as in the example above).

Writing a CFPG is very much like writing a computer program and is a very straightforward logical process by comparison with the task of writing a context-sensitive grammar. The context-free form of the core productions is an immense simplification: In writing a

context-sensitive rule for a conventional grammar, it is necessary to check that the context of the rule is such that the rule will be unambiguously selected to be applied in the desired situation and only in that situation. The facility of the programmed grammar for specifying the sequence of productions by means of the branch fields further guards against undesirable ambiguity.

Computer programs for generating strings from programmed grammars and for analyzing strings in terms of programmed grammars are discussed later in this chapter. The mechanisms embodied in these programs are not overly complicated in spite of the fact that they have been designed to operate on arbitrary CFPG's (supplied as input to the programs). Again the context-free form of the cores is largely responsible for the simplicity of the situation, since this eliminates the need to check context in determining the applicability of a production. Perhaps even more important, the action taken by a programmed grammar at each stage of a generation is much more precisely determined (less "ambiguous") than it is for a conventional grammar. To appreciate this, observe that to make the action of a programmed grammar as ambiguous as that of a conventional grammar, it would be necessary to 1) allow the productions of a programmed grammar to be applied to <u>any</u> occurrence of a variable in the current string rather than just the leftmost occurrence, and 2) place every production label in both branch fields of every production of the programmed grammar. This aspect is especially important with regard to the top-down analysis algorithm to be proposed, because it tends to make such an algorithm considerably more efficient.

9.1 Stochastic Programmed Grammars

As noted above, one reason for using programmed grammars is to gain the capability of generating and analyzing context-sensitive languages by means of a relatively simple and convenient formalism. An extension of programmed grammars will now be developed which adds the features of probabilistic grammars discussed in Chapter 8 while retaining the basic simplicity of the programmed grammar.

The most obvious feature of the programmed grammar which can be made probabilistic is the selection of the next production label from the prescribed branch field in case the field contains more than one choice (in the nonprobabilistic case, this selection is nondeterministic). By assigning branch probabilities one effectively impresses a distribution over the production set.

Definition 9.3: A stochastic programmed grammar is a 6-tuple

$$G_{spg} = (V_N, V_T, P, D, J, S)$$

where V_N, V_T, P, J, and S are as given in Definition 9.1 and D is a rule for assigning a probability measure over the productions.

Definition 9.4: Let $x \in L(G_{spg})$ be a string generated by a stochastic programmed grammar G_{spg} using the sequence of productions $r_1, r_2, \ldots, r_n$. The generation process can be represented as

$$S = Y_1 \xrightarrow{r_1} Y_2 \xrightarrow{r_2} \cdots \xrightarrow{r_n} Y_{n+1} = x.$$

The probability associated with the generation is defined to be the product of conditional probabilities

$$p(r_1)p(r_2 \mid r_1) \cdots p(r_n \mid r_1, r_2, \ldots, r_{n-1}).$$

If the string x can be generated by k distinct sequences of productions, then the probability associated with x is defined as

$$g(x) = \sum_{1}^{k} p(r_1)p(r_2 \mid r_1) \cdots p(r_n \mid r_1, r_2, \ldots, r_{n-1})$$

where the sum is over the distinct generations of x.

As in the case of stochastic phrase-structure grammars, a production probability assignment is defined to be consistent provided

$$\sum_{x \in L(G_{spg})} g(x) = 1.$$

The problem of providing a set of necessary and sufficient conditions for a production probability assignment to be consistent is greatly complicated by the branching mechanism and the general form of the core productions. A general solution to this problem will not be attempted here, although some special cases will be considered later.

If the production probability assignment is specified by assigning branching probabilities, then one can write

$$p(r_i \mid r_1, r_2, \ldots, r_{i-1}) = p(b_{r_i} \mid r_1, r_2, \ldots, r_{i-1})$$

where the right-hand term denotes the probability that after the $(i-1)^{th}$ rule is applied, a branch to rule r_i is selected. As in the case of stochastic phrase-structure grammars, this probability could depend explicitly on the preceding sequence of rules used to reach the current state of the generation. A simple but interesting case is that in which branch probabilities depend only on the current production and whether application of the current production was a

success or failure. When this is true, the following notation can be defined:

$$p(r_i \mid r_1, r_2, \ldots, r_{i-1}) = p(b_{r_i} \mid r_{i-1}, \xi_{i-1}) \triangleq p_\xi(j \mid r_{i-1})$$

where ξ is either S (success) or F (failure) and the rightmost term denotes the probability of selecting the j^{th} success or failure branch (depending on ξ) after applying rule r_{i-1}.

Definition 9.5: D is an S/F-dependent branch probability assignment for a programmed grammar provided

$$p(r_i \mid r_1, r_2, \ldots, r_{i-1}) = p_\xi(j \mid r_{i-1})$$

for all possible production sequences.

It is tempting to think of this relatively simple situation as analogous to the unrestricted production probability assignment defined earlier for conventional stochastic grammars. But the analogy is valid only in case the success and failure branch fields and branch probabilities are identical and each field contains the entire set of branch labels.

Example 9.2: The following grammar is the same as that given in Example 9.1a except that branch probabilities have been added.

$$G''_{SQ} = (V_N, V_T, P, D, J, S)$$

where

$V_N = \{S, A, B, C, D\}$

$V_T = \{a, b, c, d\}$

$J = \{1, 2, 3, 4, 5, 6, 7\}$

and P and D are given by:

Label	Core	Success Branches	Failure Branches	$p_S(j \mid r)^*$	$p_F(j \mid r)^*$
1	S → aAB	2,3	∅	α,1-α	1
2	A → aAC	2,3	∅	1-β,β	1
3	A → D	4	∅	1	1
4	C → d	5	6	1	1
5	D → bDc	4	∅	1	1
6	B → d	7	∅	1	1
7	D → bc	∅	∅	1	1

D is an S/F-dependent branch probability assignment. For this example it is easy to show that D is consistent, since direct computation yields:

$$p(abcd) = 1 - \alpha$$

$$p(a^n b^n c^n d^n) = \alpha\,\beta(1 - \beta)^{n-2}, \quad n = 2, 3, \ldots$$

and

$$\sum_{x \in L_{SQ}} g(x) = (1 - \alpha) + \sum_{n=2}^{\infty} \alpha\,\beta(1 - \beta)^{n-2}$$

$$= 1 - \alpha + \alpha(1 - \beta) \cdot \frac{1}{1-\beta} = 1.$$

This example provides an illustration that satisfying the following three conditions is evidently sufficient for an S/F-dependent branch probability assignment to be consistent:

---

*A probability value of 1 is arbitrarily assigned in those instances in which the probability value is superfluous because the branch field is null.

1. For each production r in the programmed grammar

$$\sum_j p_S(j \mid r) = \sum_j p_F(j \mid r) = 1$$

where the sums are over all branches in the specified (success, failure) field of the productions.

2. Every terminating generation produces a terminal string (a string having no nonterminals).

3. Every generation terminates with probability 1.

In practice, the first condition is easy to meet and the second and third conditions will normally be satisfied if there are no logical flaws in the formulation ("programming") of the grammar. The third condition is the hardest to guarantee: There must be no "trapping" loops in the grammar. The following fragment of a stochastic programmed grammar shows an example of a simple situation which can lead to violation of this condition.

Label	Core	Success Branches	...	$p_S(j \mid r)$	...
⋮	⋮	⋮		⋮	
i	A → AA	i,j	...	α,1-α	...
j	A → a	i,j	...	α,1-α	...
⋮	⋮	⋮		⋮	

Assume there is a nonzero probability that the grammar will produce at least one A to be rewritten by production i or j with probability α or 1-α, respectively. Then it follows from an elementary result

in the theory of branching processes ([62], pp. 4-9) that α must be less than 1/2 in order to guarantee that the generation will terminate.

Unfortunately the production loops which violate the third condition are usually much harder to detect than the one in this simple example; and the conditions as stated do not suggest a general procedure for testing an arbitrary stochastic programmed grammar for consistency. Of course, if the language generated contains a finite number of strings, the test can consist of direct computation; and as the previous example illustrates, this may also be true even if the language is infinite.

As the final remark of this section, it seems hardly necessary to point out that stochastic programmed grammars are computationally or generatively no more powerful than their nonprobabilistic counterparts. Both types generate the same class of languages. However, it is hoped that the stochastic generalization will lead to grammars which can model the probabilistic aspects of noisy "real world" situations (see the example in Section 9.3).

9.2 Languages with Tails

Definition 9.6 [75]: Let L_a be a language over the set of terminals V_T and let L_b be a language over $V_T \cup \{s,t\}$ where $s,t \notin V_T$, such that

$$L_b = \{xst^m \mid x \in L_a \text{ and } m \text{ depends on } x\}.$$

Then L_b is said to be of the form L_a with tails. The x portion will be referred to as the body of the string, s as the tail delimiter, and t^m as the tail.

The relationship of CFPG's and languages with tails is interesting.

Theorem 9.2 [76]: If L is a recursively enumerable language, there exists a CFPG language of the form L with tails.

This is a useful result because it further broadens the class of languages which can be dealt with under the convenient formalism of context-free programmed grammars. And in general, the tail is found to be a convenient place to perform any bookkeeping functions necessary for the generation of the language. This will be illustrated in the example presented in the next section.

9.3 A Generator for Stochastic CFPG Languages

It is a relatively simple matter to write a computer program to simulate the language generation process defined by any particular CFPG. For experimental or developmental purposes, however, it is convenient to have at hand a program which uses any given CFPG to generate strings. The generator program described in this section generates strings in a stochastic manner using any specified stochastic CFPG (read in as "data") and displays a pictorial interpretation of the generation process on a digital display screen. The generation and display capabilities permit the user to observe the programmed grammar at work, to interactively improve the grammar, and to qualitatively assess its stochastic properties.

The program, written for the Digital Equipment Corporation PDP-9 with Model 339 Graphical Display, is diagrammed in Figure 9.1.

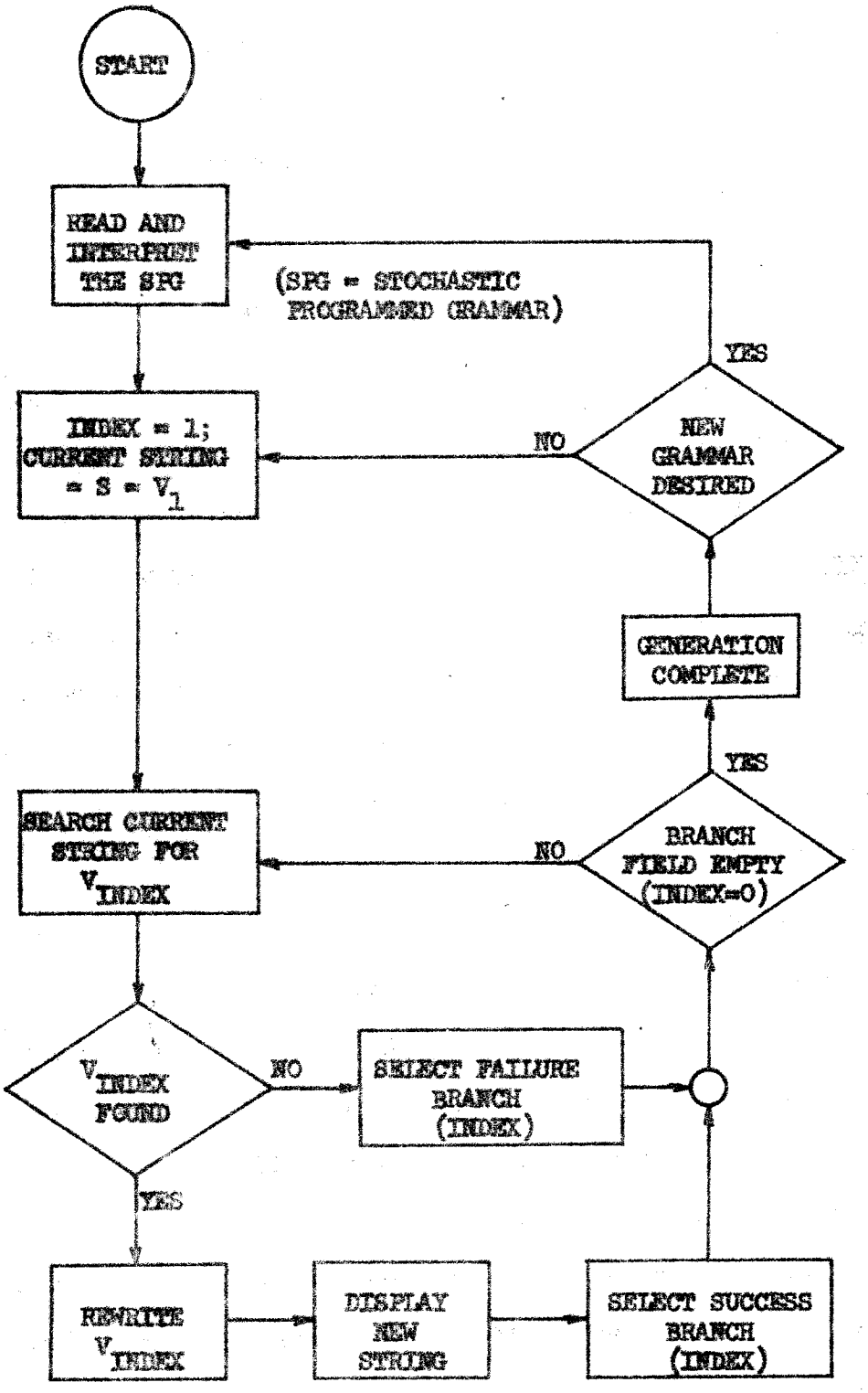


Figure 9.1 Generator Program for CFPG Languages

The main routine serves as the control program and applies the core productions to the current sentential form. A grammar input subroutine reads the stochastic CFPG from the selected input medium, interprets it (a convenient input format is used which must be converted to the form operated on by the program), and sets up the necessary list structures (Figure 9.2) which have been formulated for convenient manipulation of arbitrary CFPG's. The branch selection subroutine makes a probabilistic choice of the next production label from the labels in the appropriate branch field. A random number generator is utilized for this purpose. Finally the display subroutine interprets the current sentential form and maintains a display file for generating the graphical display. This routine is the least well defined part of the program since its exact nature depends entirely on the user-defined semantic interpretation of the strings generated by the grammar.

Example 9.3: It was proposed to develop a grammar for "noisy squares" to model a physical process, namely the drawing of squares on, say, a CRT input device or a RAND tablet. The following assumptions were made to keep the model simple but interesting.

1. The squares were to have their side length geometrically distributed with mean length $1/\alpha$ ($\alpha < 1$).

2. The squares were to be drawn horizontal/vertical and the operator was assumed skillful enough so that the "noise" could be considered principally the quantization error resulting from input through a digital or finite-state device.

Pointer to Start of Prod'n 1 $(P_1)^*$	Length of Right Side of Prod'n 1 $(\lvert R_1 \rvert)^{**}$	Length of Success Field of Prod'n 1 $(\lvert S_1 \rvert)^{**}$	Length of Failure Field of Prod'n 1 $(\lvert F_1 \rvert)^{**}$
Pointer to Start of Prod'n 2 $(P_2)^*$	etc.		

*Pointer to Memory

**Length Value

Production Pointer Stack

Left Side of Prod'n 1
Right Side of Prod'n 1*
Success Field for Prod'n 1*
Failure Field for Prod'n 1*
Success Branch Probabilities for Prod'n 1*
Failure Branch Probabilities for Prod'n 1*
Left Side of Prod'n 2

*Occupies as Many Memory Locations as Required by the Production

Production Stack

Figure 9.2 Program Storage Configuration For Stochastic CFPG's

3. The "target aiming" effect, which would come into play as a square were about to be closed and would assist in ensuring closure, was ignored.

The simplifying assumptions could be relaxed and the model made arbitrarily complex. However, the example as defined serves to illustrate that:

1. a relatively complex context-sensitive language can be generated by a CFPG;

2. a "noisy" physical situation can be modeled by a stochastic programmed grammar;

3. a general program for simulating the operation of programmed grammars can aid in the interactive development of grammars intended to model physical situations.

A language with tails was used so that the tails could contain necessary counters.* The first stochastic CFPG formulated to generate the noisy squares exhibited an obvious tendency to "ring" after noise appeared; i.e., the noise was self-sustaining rather than having a tendency to die out (Figure 9.3). This characteristic was found to be inherent in the grammar: The restoring mechanism which was intended to bring deviations back to the straight line continued to operate in the same direction after the line was reached, thereby resulting in overshoot.

*As formulated, the grammar does not bother to convert nonterminals in the tail to terminal symbols, a step of no practical significance.

The first revision of the original grammar appeared to have over-compensated for the ringing (Figure 9.4). This was easily adjusted, however, by modifying the branch probabilities of a few productions. The final version generates the sort of noisy squares desired (Figure 9.5); the grammar appears in Table 9.1. "Reasonable" values were selected for the branch probabilities in the model. If a hand-drawn "training set" were available for analysis in terms of the grammatical model, these probabilities could be more closely tuned to the true physical situation.

9.4 Analysis of Strings Generated by Programmed Grammars

This section presents a method for analyzing strings to determine whether they belong to the language generated by a given CFPG. If the grammar is stochastic, the probability of generation is computed for each string found to belong to the language.

The analyzer is a top-down parsing algorithm consisting roughly of the generation algorithm of the previous section plus a back-tracking algorithm which together can systematically trace out the generation tree associated with the given CFPG. In the process, every valid generation of a string to be analyzed is found and reported. The concept is similar to the idea suggested by Chartes and Florentin for the analysis of context-free languages [77], although the resulting analyzer for CFPG languages is necessarily quite unlike their algorithm.

The only restriction on the CFPG is that it contain no cycles of nonterminals which do not increase the length of the string being

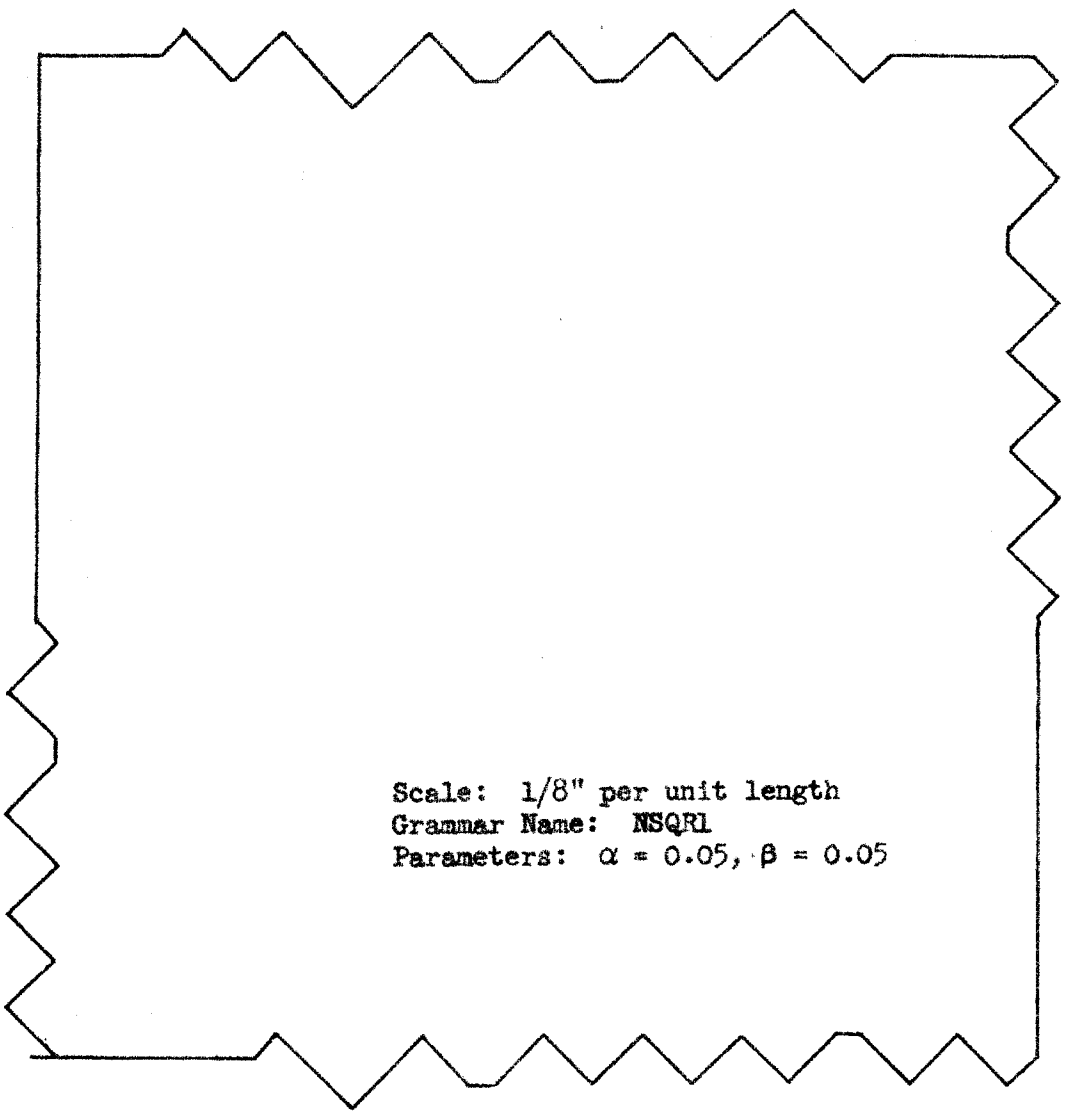


Figure 9.3 A "Ringing" Noisy Square

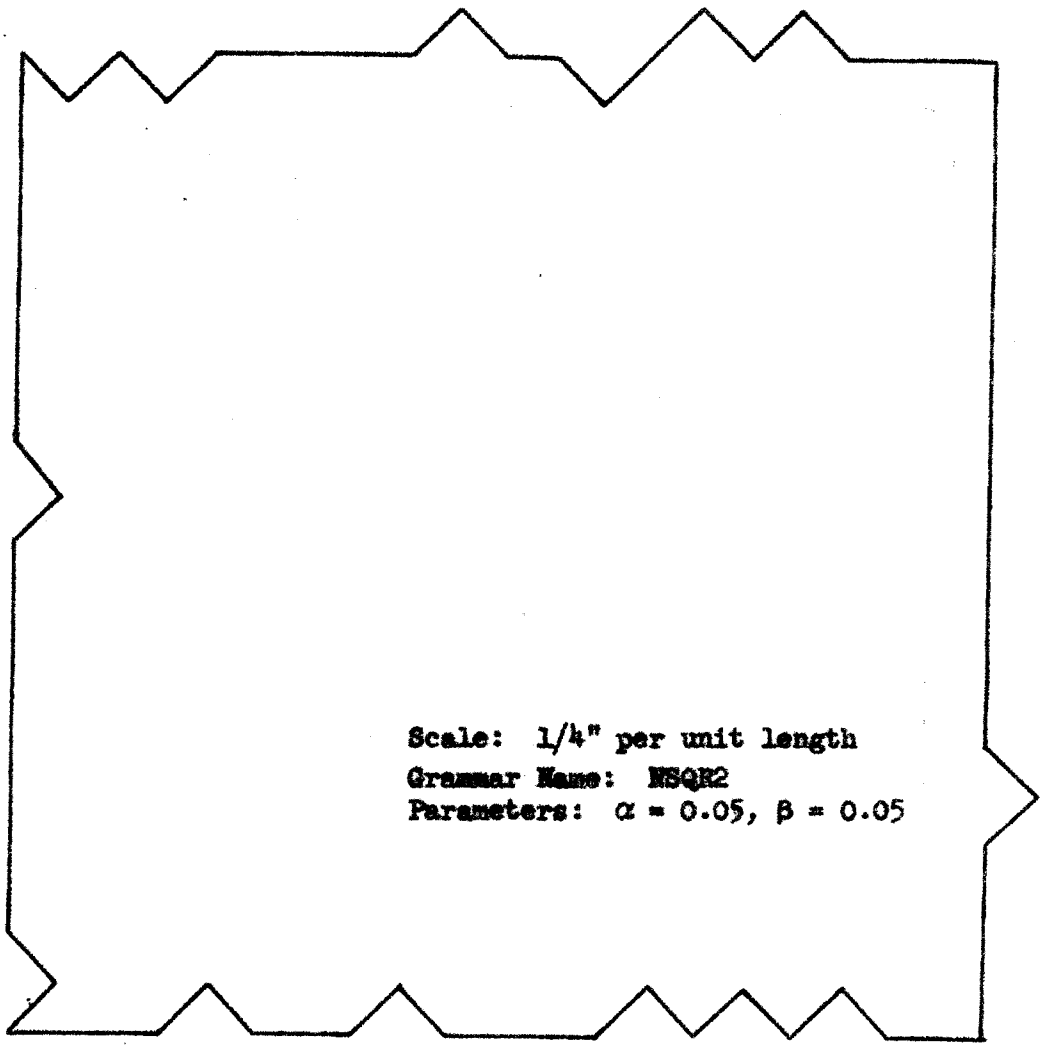


Figure 9.4 A Noisy Square with Ringing Removed

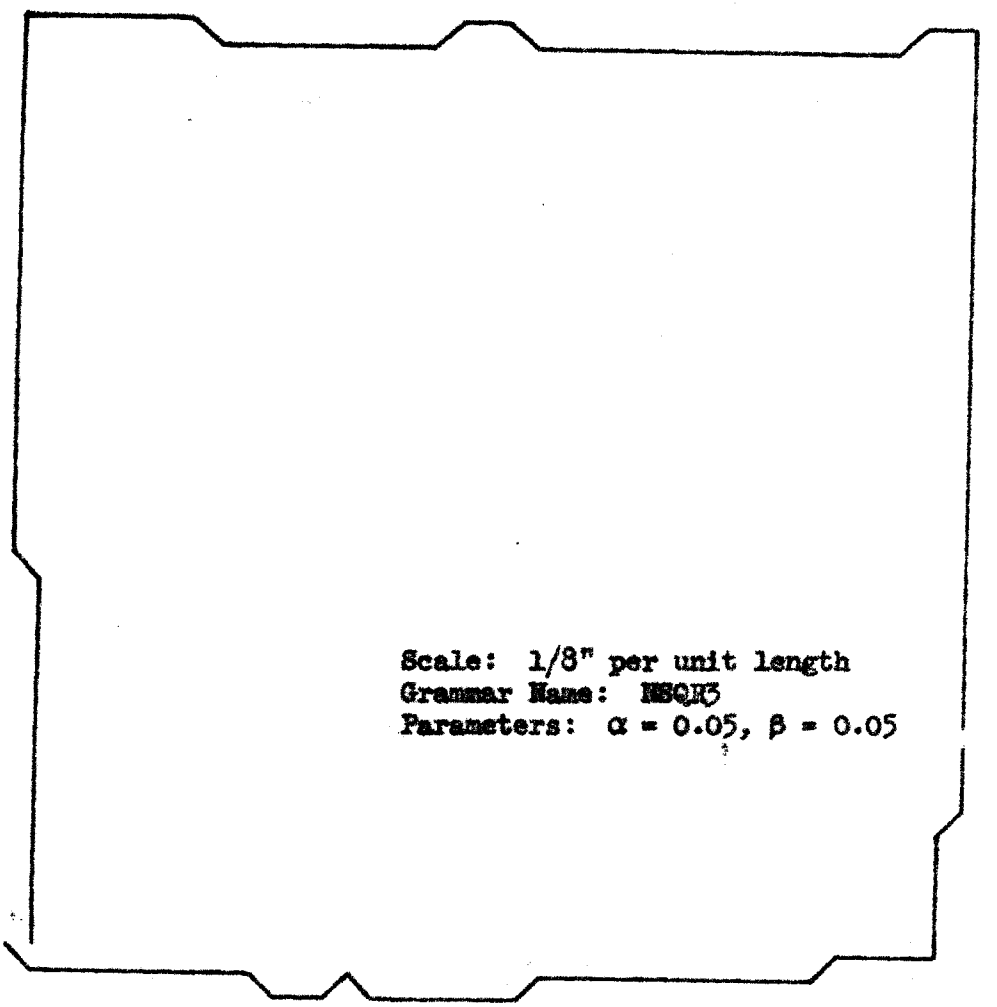


Figure 9.5 A Noisy Square from "Improved" Stochastic CFPG

Table 9.1 NSQR3: An "Improved" Grammar For Noisy Squares

$$G_{NSQR3} = (V_N, V_T, P, D, J, S)$$

where

$$V_N = \{S, F, X, Q, A, M, P, D, E, R, T, B, V\}$$

$$V_T = \{-, 0, +, *, \$\}$$

$$J = \{1, 2, \ldots, 37\}$$

and P and D are given by:

Label	Core	Success Branches	Failure Branches	$p_S(j \mid r)^\dagger$	$p_F(j \mid r)^\dagger$
1	S → FXVQ	2,4,7	∅	$1-2\alpha,\alpha,\alpha$	1
2	F → 0FA	3	∅	1	1
3	X → XXX	10	∅	1	1
4	F → +FA	5	∅	1	1
5	M → XXX	10	6	1	1
6	X → PXX	10	∅	1	1
7	F → -FA	8	∅	1	1
8	P → XXX	10	9	1	1
9	X → MXX	10	∅	1	1
10	P → P	2,4,7,12	11	v_1,v_2,v_3,β	1
11	M → M	2,4,7,12	2,4,7,12	v_1,v_3,v_2,β	v_4,v_3,v_3,β
12	F → *F	13	∅	1	1
13	Q → R	16	14	1	1
14	R → S	16	15	1	1
15	S → T	16	36	1	1
16	B → A	16	17	1	1
17	A → B	18	20	1	1
18	E → D	17	19	1	1
19	V → DV	17	∅	1	1

$^\dagger v_1 = (1-\alpha-\alpha^2)(1-\beta)$; $v_2 = \alpha^2(1-\beta)$; $v_3 = \alpha(1-\beta)$; $v_4 = (1-2\alpha)(1-\beta)$

Table 9.1, cont.

Label	Core	Success Branches	Failure Branches	$p_S(j \mid r)$	$p_F(j \mid r)$
20	P → X	21	22	1	1
21	D → E	20	ϕ	1	1
22	M → X	23	25,26,29	1	$1-2\alpha,\alpha,\alpha$
23	E → D	22	24	1	1
24	V → DV	22	ϕ	1	1
25	F → OF	32	ϕ	1	1
26	F → +F	27	ϕ	1	1
27	M → X	32	28	1	1
28	X → P	32	ϕ	1	1
29	F → -F	30	ϕ	1	1
30	P → X	32	31	1	1
31	X → M	32	ϕ	1	1
32	D → E	33	ϕ	1	1
33	D → D	34	12	1	1
34	P → P	25,26,29	35	$1-\alpha^2-\alpha,\alpha^2,\alpha$	1
35	M → M	25,26,29	25,26,29	$1-\alpha^2-\alpha,\alpha,\alpha^2$	$1-2\alpha,\alpha,\alpha$
36	T → $	37	ϕ	1	1
37	F → F	ϕ	ϕ	1	1

Semantic interpretation of terminals:

0: unit segment in direction of current side
+: unit segment with slope + 1 relative to current side
-: unit segment with slope - 1 relative to current side
*: corner (change direction by + 90°)
$: tail delimiter (final string only)

Example: The noisy square shown in Figure 9.5 is the pictorial representation of the string:

-000000000-00+-000000+00000000000+0000*

00000-0000000000000000000000000000000*

00+0000000000000-00+000000000-0000000*

00000000000000000000000+00000000000000*$ (tail omitted)

generated (e.g., $A \xrightarrow{r_1} B$, $B \xrightarrow{r_2} C$, $C \xrightarrow{r_3} A$, followed by another application of r_1). Practically speaking this is a very minor restriction, but it guarantees that the analysis procedure will always terminate. It ensures that every path down the generation tree produces a progressively longer sentential form (by definition, no production of a CFPG may generate the null string). Since any candidate analysis is terminated if the length of the current sentential form exceeds the length of the string being analyzed, and since there are at most a finite number of such candidate analyses, the procedure must eventually halt.

To implement the backtracking algorithm, the grammar is modified (automatically by the analyzer) in a manner which, in effect, embeds the CFPG language in a bracket language [78]. In the case of context-free phrase-structure grammars, a bracket language has the feature that every string contains within itself the complete specification of the way in which it was generated. In the CFPG case, each string contains a history of the successful productions used to generate it. When the analyzer is backtracking over a step which involves the successful application of a production, the bracket configuration indicates which portion of the sentential form was produced by that step and must therefore be reduced to a single nonterminal.

Example 9.4: $L(G) = \{a^n b^n c^n \mid n \geq 1\}$ is generated by the following CFPG.

$$G = (V_T, V_N, P, J, A)$$

where

$V_N = \{A, B, C\}$

$V_T = \{a, b, c\}$

$J = \{1, 2, 3, 4, 5\}$

and P consists of 5 productions:

Label	Core	Success Branch	Failure Branch
1	A → aBC	2,4	∅
2	B → aBB	3	∅
3	C → CC	2,4	∅
4	B → b	4	5
5	C → c	5	∅

Figure 9.6 is a schematic of the analysis of the string 'abc' in terms of this CFPG. The notation used is as follows.

1. For any nonterminal $X \in V_T$, $X_i(\gamma)X_i$ indicates that the string γ was generated from the nonterminal X which was rewritten as γ on the i^{th} step.

2. A downward arrow indicates a generative step; an upward arrow indicates backtracking.

3. The branch labels have the form $\xi_i(j) = r_k$, where ξ is S (success) or F (failure), subscript i indicates application of the i^{th} production, j indicates selection of the j^{th} branch in the ξ branch field, which is the branch to the production labelled r_k. By definition: the first (top-most) label is $S_o(1) = 1$; $\xi_i(j) = \emptyset$ indicates termination of a path down the tree, which constitutes a successful analysis if the current sentential form is identical with the string

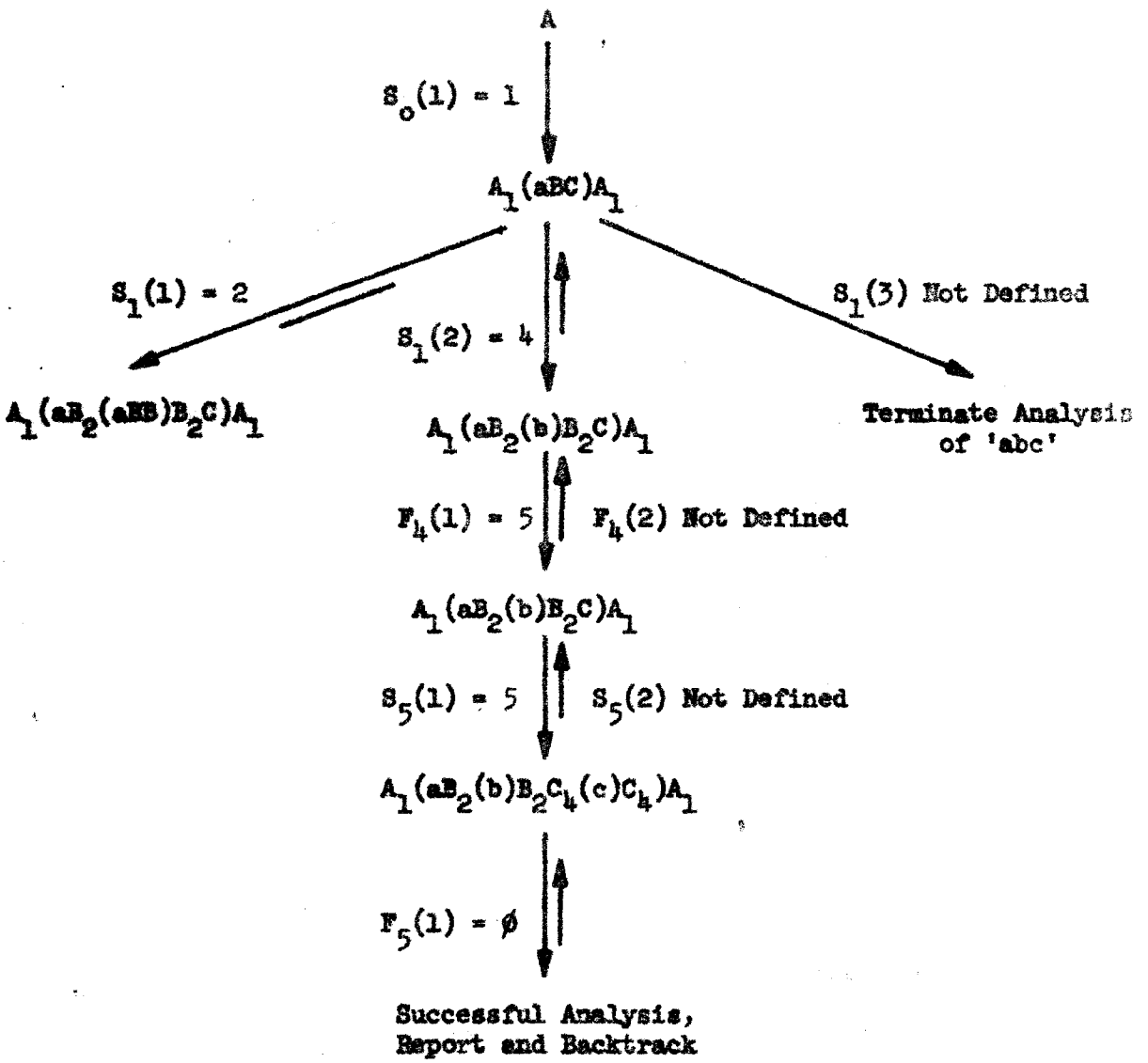


Figure 9.6 Analysis of 'abc' in Terms of the Grammar of Example 9.4

being analyzed; the analysis of the string is complete when $\xi_1(j)$ is undefined (or null) for i = 1 and some j.

Notice that the analyzer must continue even after a successful analysis is produced since, in general, it is not known a priori whether the grammar is unambiguous and it is desired to produce all possible analyses.

The analyzer program, written in FORTRAN for the IBM System/360 Model 44, is diagrammed in Figure 9.7. The program uses the same input subroutine written for the generation program for reading and interpreting the CFPG and setting up the grammar-associated list structures. One additional list, containing a history of all productions used (whether or not successful), is maintained by the program. The formats used for this list and for the list which contains the current sentential form are shown in Figure 9.8.

When the grammar used for the analysis is a stochastic CFPG with an S/F-dependent branch probability assignment, the report of each successful analysis includes the associated probability of generation.

The analyzer has demonstrated its capability with a range of context-free programmed grammars including the "noisy squares" grammar discussed in the previous section.

Example 9.5: Figure 9.9 shows an example of the computer output produced by the analyzer program given the "noisy squares" grammar (NSQR3) as input. The string supplied for analysis in this case is a valid noisy square and the associated probability is printed by the

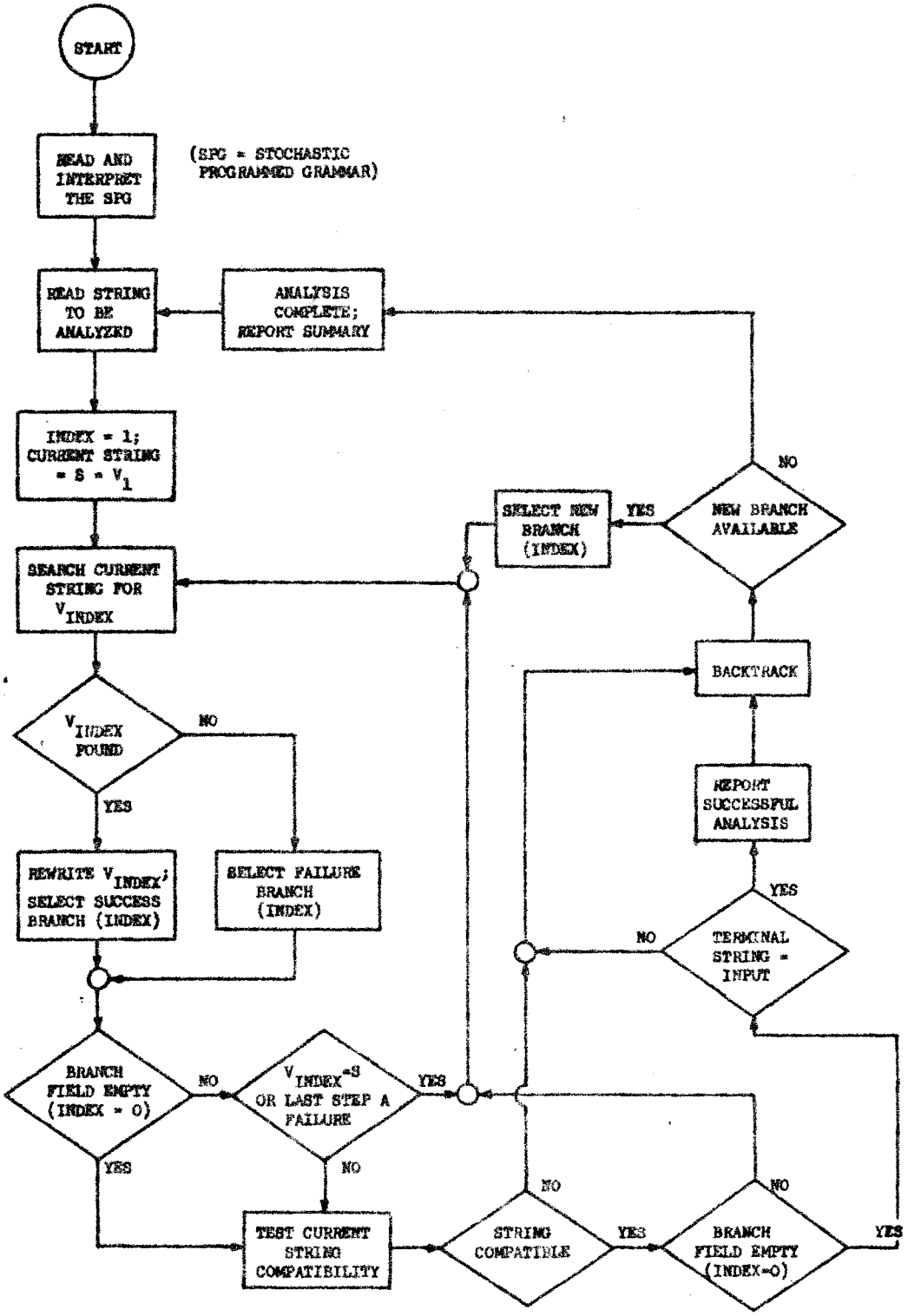
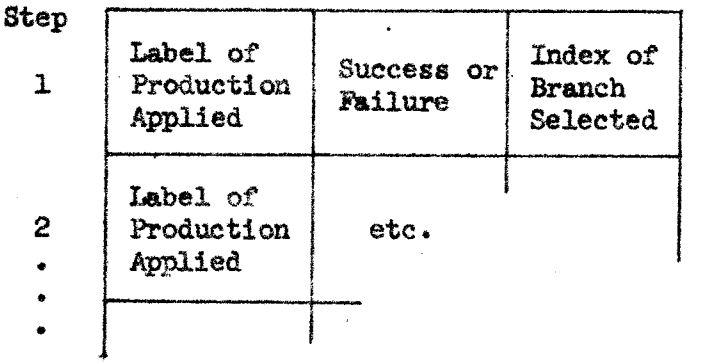
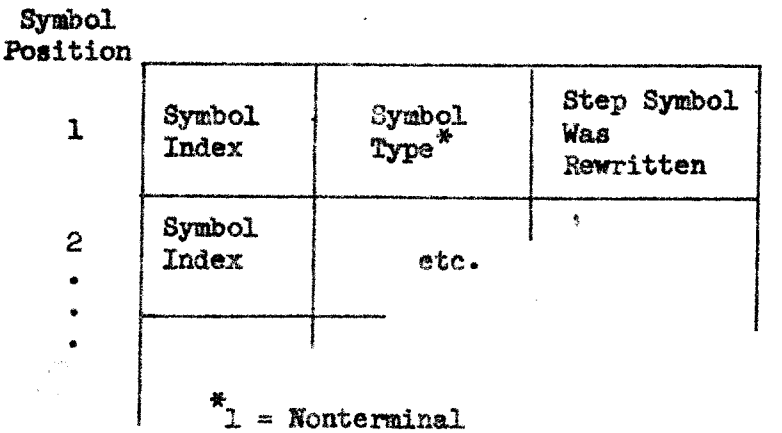


Figure 9.7 Analyzer Program for CFPG's



Generation Record Stack



Current Sentential Form

Figure 9.8 Special Stacks Used by the CFPG Analyzer

```
NEW GRAMMAR....

VARIABLES  S F X Q A M P D E R T B V
TERMINALS  - 0 + * $
PRODUCTIONS....
```

LABEL		CORE	SUCCESS BRANCHES	FAILURE BRANCHES	BRANCH PROBABILITIES (X 100) SUCCESS	FAILURE
1.	S	--* FXVQ	2 4 7	0	90 5 5	100
2.	F	--* OFA	3	0	100	100
3.	X	--* XXX	10	0	100	100
4.	F	--* +FA	5	0	100	100
5.	M	--* XXX	10	6	100	100
6.	X	--* PXX	10	0	100	100
7.	F	--* -FA	8	0	100	100
8.	P	--* XXX	10	9	100	100
9.	X	--* MXX	10	0	100	100
10.	P	--* P	2 4 7 12	11	88 2 5 5	100
11.	M	--* M	2 4 7 12	2 4 7 12	88 5 2 5	85 5 5 5
12.	F	--* *F	13	0	100	100
13.	Q	--* R	16	14	100	100
14.	R	--* S	16	15	100	100
15.	S	--* T	16	16	100	100
16.	B	--* A	16	17	100	100
17.	A	--* B	18	20	100	100
18.	E	--* D	17	19	100	100
19.	V	--* DV	17	0	100	100
20.	P	--* X	21	22	100	100
21.	D	--* E	20	0	100	100
22.	M	--* X	23	25 26 29	100	90 5 5
23.	E	--* D	22	24	100	100
24.	V	--* DV	22	0	100	100
25.	F	--* DF	32	0	100	100
26.	F	--* +F	27	0	100	100
27.	M	--* X	32	28	100	100
28.	X	--* P	32	0	100	100
29.	F	--* -F	30	0	100	100
30.	P	--* X	32	31	100	100
31.	X	--* M	32	0	100	100
32.	D	--* E	33	0	100	100
33.	D	--* D	34	12	100	100
34.	P	--* P	25 26 29	35	92 3 5	100
35.	M	--* M	25 26 29	25 26 29	92 5 3	90 5 5
36.	T	--* $	37	0	100	100
37.	F	--* F	0	0	100	100

```
ANALYZE STRING  0+-$-00*0000*+-0*

SUCCESSFUL ANALYSIS
S  1 (      F  2 (    0  0       F  6 (    +  0       F 10 (    -  0       F 14 (    *  0       F 29 (
-  0        F 36 (    0  0       F 41 (    0  0       F 44 (    *  0       F 63 (    0  0       F 68 (
0  0        F 73 (    0  0       F 78 (    0  0       F 81 (    *  0       F 98 (    +  0       F104 (
-  0        F110 (    0  0       F113 (    *  0       F118 (    0  0       F118 )    F113 )    F110 )
F104 )      F 98 )    F 81 )    F 78 )    F 73 )    F 68 )    F 63 )    F 44 )    F 41 )    F 36 )
F 29 )      F 14 )    A 17 )    A 67 )    A 51 )    A 20 )    A 89 )    B 0  )    A 89 )    B 85 )
A 51 )      B 47 )    A 17 )    B 67 )    F 10 )    B 48 )    A 93 )    B 86 )    A 91 )    B 0  )
A 91 )      B 86 )    A 53 )    A 48 )    A 20 )    A 20 )    A 93 )    B 49 )    A 55 )    B 97 (
A 93 )      B 0  )    A 93 )    B 67 )    A 55 )    B 48 )    A 55 )    F 2  )    A 3  (    B 8  (
P  9 (      P 11 (    X 31 (    M 35 (    M 40 )    M 59 (    X100 (    P103 (    P105 (    X  0 (
P105 )      P103 )    X100 )    M 59 )    M 40 (    M 55 )    X 31 (    X  0 (    X  0 )    P  1 (
P  9 )      X  0 (    X  0 )    X  8 (    X  0 )    X  0 )    X  3 (    V 19 (    D 32 (    E 42 (
D 64 (      E 90 (    D101 (    E  0 (    D101 (    E 90 (    D 64 (    E 52 (    D 32 )    E 22 (
D 33 (      D 37 (    E 54 (    D 65 (    D 69 (    E 92 (    D102 (    D106 (    E  0 (    D106 (
D102 (      E 92 (    D 69 (    D 65 (    E 54 (    D 37 (    B 33 )    V 25 (    D 38 (    D 42 (
E 96 (      D 79 (    B 74 (    E 94 (    D107 (    D111 (    E  0 (    D111 (    D107 )    E 94 (
D 74 (      D 70 (    E 56 (    D 42 (    D 38 (    V 61 (    D 75 (    D 79 (    E  0 (    D 79 (
D 75 (      V  0 (    V 61 (    V 25 (    V 22 (    V 29 (    Q 15 (    R 46 (    S 84 (    T117 (
$  0        T117 )    S 84 )    R 46 )    Q 15 )    S  1 )

ASSOCIATED PROBABILITY =  0.70280-08

ANALYSIS COMPLETE      1 SUCCESSFUL GENERATION(S)
```

Figure 9.9 Output from the Stochastic CFPG Analyzer

analyzer. The output following "SUCCESSFUL ANALYSIS" is a listing of the "Current Sentential Form" stack generated in the course of reaching the successful analysis. It may be interpreted as follows. The listing is read left-to-right, top-to-bottom in the usual manner. The first symbol of each triple is either a variable or a terminal. The second symbol indicates the step in the generation at which the symbol was rewritten (zero for terminals and for final tail symbols not converted to terminals). The third symbol indicates whether the triple represents a left bracket ("("), a right bracket (")"), the tail delimiter ("$"), or a terminal or final tail variable (blank). For example, the first triple represents the left bracket produced by rewriting the start symbol S on the fist step.

The listing of the final sentential form provides a complete record of the syntactic variables from which the terminal string is derived and also a complete history of the "ancestry" of each variable extending back inevitably to the start symbol. Note, however, that this list in not a complete history of the generation process: the "failure" steps are not indicated. A complete description of the generation process is available from the "Generation Record Stack" (Figure 9.8), if desired.

9.5 Summary

The principal goal of Chapters 8 and 9 has been to establish some framework for a probabilistic model for linguistic pattern recognition. One of the most serious obstacles to the development of practical linguistic recognition models is the lack of syntactic

analysis procedures efficient enough to process in a practical manner the highly complex pattern structures toward which such procedures will be directed. Two approaches to the development of efficient syntactic analyzers have been considered here.

In Chapter 8 some basic efforts have been made to determine to what extent the relationships between automata and formal languages may be exploited in this respect, since automata are conceptually simple recognition devices. Attention has been given to the probabilistic generalization of these relationships, particularly with regard to finite automata and finite-state languages. The computation of probability-valued functions by stochastic automata has been defined, and it has been shown that stochastic finite automata can recognize and compute the probability of generation of any string generated by a stochastic finite-state grammar which has an unrestricted and consistent production probability assignment. It has been found that the concept of recognition with isolated cut-points (and nonzero cut-points in general) may be of little value for recognizing stochastic pattern languages containing an infinite number of patterns, because such recognition is incompatible with computation of the probability of generation of the patterns. A similar conclusion has been reached with respect to recognition by automata under a maximum likelihood criterion, as recently proposed in the literature. Some consideration has been given to the question of whether stochastic automaton recognizers should actually be constructed or whether it would be preferable to simply use the automata theory to derive a

computational model for syntactic pattern recognition.

In Chapter 9 attention has been focused on developing a special stochastic grammar which may be better suited to practically describing and analyzing patterns than the more familiar forms of phrase-structure grammars. The recently formulated context-free programmed grammar--a relatively simple formalism for generating context-sensitive languages (and even type 0 languages "with tails")--has been adopted as a foundation and generalized by adding a probabilistic mechanism. An attempt has been made to demonstrate the feasibility of this approach to probabilistic pattern generation and analysis by realizing these processes in terms of computer programs and applying the programs to a class of noisy patterns. It is hoped that the results will encourage further work in this direction.

CHAPTER 10

RECOMMENDATIONS AND CONCLUDING REMARKS

Some variations of the proposed method for nonparametric pattern recognition should be investigated. Generalization of the smoothing technique to utilize a smoothing matrix not limited to be diagonal in form has already been suggested. Other functional forms, possibly better suited to specific types of data distributions, should be tried as the "kernel" of the estimator (the reasons for using the exponential kernel were cited in Chapter 3). This could lead in particular to improved behavior of the truncated polynomial approximation when use of the approximation were deemed necessary.

Linguistic pattern recognition is still in its infancy and seeking profitable directions in which to grow. Two possible directions have been suggested here but each will require considerable development before practical pattern recognition systems can be realized based on these ideas. Most of the results of Chapter 8 concern approaches which should probably not be tried where infinite pattern languages are involved, but the value of automaton-like recognizers for dealing with finite languages and finite approximations of infinite languages remains to be explored. The programmed grammar is just one interesting variation on the conventional phrase-structure grammar. Other specialized grammars may be

formulated, possibly better suited to a probabilistic generalization. To be useful however, they must be amenable to effective and efficient syntactic analysis procedures. In this respect, it would be interesting to consider the development of a special purpose hardware implementation of the programmed grammar analyzer proposed in Chapter 9.

In conclusion, throughout the course of the research reported herein, the author has tried to achieve a useful balance of theory and engineering ingenuity so as to arrive at some new methods for dealing with the increasingly complex nature of contemporary pattern recognition problems. To the extent that such a reasonable balance has been struck, the readers who are theoreticians will be able to accept as plausible the ideas presented (with supporting empirical evidence) and perhaps pursue them further, while the readers with an engineering orientation will be tempted to utilize the proposed methods in the solution of their pattern recognition problems.

LIST OF REFERENCES

1. D. G. Keehn, "Learning the mean vector and covariance matrix of Gaussian signals in pattern recognition," Stanford Electronics Lab., Stanford, Calif., Tech. Rept. TR 2003-6, February 1963.

2. N. J. Nilsson, Learning Machines. New York: McGraw-Hill, 1965.

3. K. S. Fu, Sequential Methods in Pattern Recognition and Machine Learning. New York: Academic, 1968.

4. P. M. Lewis, "The characteristic selection problem in recognition systems," IRE Trans. Information Theory, vol. IT-8, pp. 171-178, February 1962.

5. T. Marill and D. M. Green, "On the effectiveness of receptors in recognition systems," IEEE Trans. Information Theory, vol. IT-9, pp. 11-17, January 1963.

6. K. S. Fu, P. J. Min, and T. J. Li, "Feature selection in pattern recognition," IEEE Trans. Systems Science and Cybernetics, vol. SSC-6, January 1970.

7. G. Sebestyen, Decision-Making Processes in Pattern Recognition. New York: Macmillan, 1962.

8. D. F. Specht, "Generation of polynomial discriminant functions for pattern recognition," Stanford Electronics Lab., Stanford, Calif., Tech. Rept. 6764-5, May 1966.

9. G. R. Cooper and J. A. Tabaczynski, "Estimation of probability density and distribution functions," School of Elec. Engrg., Purdue University, Lafayette, Ind., Tech. Rept. TR-EE 65-15, August 1965.

10. S. S. Wilks, Mathematical Statistics. New York: Wiley, 1962.

11. M. Rosenblatt, "Remarks on some nonparametric estimates of a density function," Ann. Math. Stat., vol. 27, pp. 832-837, 1956.

12. P. Whittle, "On the smoothing of probability density functions," J. Royal Statistical Soc., ser. B, vol. 20, no. 2, pp. 334-343, 1958.

13. E. Parzen, "On estimation of a probability density function and mode," Ann. Math. Stat., vol. 27, pp. 1065-1076, September 1962.

14. G. S. Watson and M. R. Leadbetter, "On the estimation of the probability density," Ann. Math. Stat., vol. 34, pp. 480-491, 1963.

15. V. K. Murthy, "Nonparametric estimation of multivariate densities with applications," Douglas Missile and Space Systems Division, Douglas Aircraft Co., Santa Monica, Calif., paper 3490, June 1965; also in Multivariate Analysis. P. R. Krishnaiah, Ed., New York: Academic, 1966.

16. Ya. Z. Tsypkin, "Use of the stochastic approximation method in estimating unknown distribution densities from observations," Automation and Remote Control, vol. 27, no. 3, pp. 432-434, 1966.

17. M. A. Aizerman, E. M. Braverman, and L. I. Rozonoer, "The probability problem of pattern recognition and learning and the method of potential functions," Automation and Remote Control, vol. 25, pp. 1175-1193, September 1964.

18. R. R. Lemke, "On the application of the potential function method to pattern recognition and system identification," School of Elec. Engrg., Purdue University, Lafayette, Indiana, Tech. Rept. TR-EE 68-8, April 1968.

19. K. S. Fu, Y. T. Chien, Z. J. Nikolic, and W. G. Wee, "On the stochastic approximation and related learning techniques," School of Elec. Engrg., Purdue University, Lafayette, Indiana, Tech. Rept. TR-EE 66-6, April 1966.

20. C. C. Blaydon, "On a pattern classification result of Aizerman, Braverman, and Rozonoer," IEEE Trans. Information Theory, vol. IT-12, pp. 82-83, January 1966.

21. R. L. Kashyap and C. C. Blaydon, "Estimation of probability density and distribution functions," School of Elec. Engrg., Purdue University, Lafayette, Indiana, Tech. Rept. TR-EE 67-14, August 1967.

22. R. Kronmal and M. Tarter, "The estimation of probability densities and cumulatives by Fourier series methods," J. Amer. Statistical Association, vol. 63, pp. 925-952, September 1968.

23. D. O. Loftsgaarden and C. P. Quesenberry, "A nonparametric estimate of a multivariate density function," Ann. Math. Stat., vol. 36, pp. 1049-1051, 1965.

24. E. A. Patrick and F. K. Bechtel, "A nonparametric recognition procedure with storage constraint," School of Elec. Engrg., Purdue University, Lafayette, Ind., Tech. Rept. TR-EE 69-24, August 1969.

25. K. S. Fu and E. G. Henrichon, Jr., "On nonparametric methods for pattern recognition," School of Elec. Engrg., Purdue University, Lafayette, Indiana, Tech. Rept. TR-EE 68-19, August 1968.

26. H. Cramèr, Mathematical Methods of Statistics. Princeton, N. J.: Princeton University Press, 1946.

27. G. H. Ball, "Data analysis in the social sciences: What about the details?" Proc. Fall Joint Computer Conference, December 1965.

28. R. L. Mattson and J. E. Dammann, "A technique for determining and coding subclasses in pattern recognition problems," IBM Journal of Research and Development, vol. 9, pp. 294-302, July 1965.

29. T. W. Anderson, An Introduction to Multivariate Statistical Analysis. New York: Wiley, 1958.

30. G. H. Ball and D. J. Hall, "ISODATA, a novel method of data analysis and pattern classification," Stanford Research Institute, Menlo Park, Calif., April 1965.

31. R. L. Grimsdale, F. H. Sumner, C. J. Tunis, and T. Kilburn, "A system for the automatic recognition of patterns," Proc. Institution of Electrical Engineers, vol. 106, part B, pp. 210-221, March 1959.

32. R. S. Ledley, L. S. Rotolo, T. J. Golab, J. D. Jacobsen, M. D. Ginsburg, and J. E. Wilson, "FIDAC: Film input to digital automatic computer and associated syntax-directed pattern recognition programming system," Optical and Electro-Optical Information Processing Systems. J. Tippet, D. Beckowitz, L. Clapp, C. Koester, and A. Vanderburgh, Jr., Eds., Cambridge, Mass.: M. I. T. Press, 1965.

33. A. C. Shaw, "The formal description and parsing of pictures," Stanford Linear Accelerator Center, Stanford University, Stanford, Calif., SLAC Rept. 84, March 1968.

34. K. S. Fu and P. H. Swain, "On syntactic pattern recognition," presented at the Third International Symposium on Computer and Information Science (COINS-69), Bal Harbour, Fla., December 1969.

35. H. Freeman, "On the encoding of arbitrary geometric configurations," IRE Trans. Electronic Computers, vol. EC-10, pp. 260-268, 1961.

36. P. J. Knoke and R. G. Wiley, "A linguistic approach to mechanical pattern recognition," Proc. IEEE Computer Conference, pp. 142-144, September 1967.

37. T. Pavlidis, "Analysis of set patterns," Information Science and Systems Lab., Dept. of Elec. Engrg., Princeton University, Princeton, N. J., Tech. Rept. 25, 1968.

38. A. Rosenfeld and J. P. Strong, "A grammar for maps," presented at the Third International Symposium on Computer and Information Science, Bal Harbour, Fla., December 1969.

39. M. Eden, "Handwriting and pattern recogniton," IRE Trans. Information Theory, vol. IT-8, pp. 160-166, 1963.

40. T. Pavlidis, "Linguistic analysis of waveforms," presented at the Third International Symposium on Computer and Information Science, Bal Harbour, Fla., December 1969.

41. R. H. Anderson, "Syntax-directed recognition of hand-printed two-dimensional mathematics," Ph.D. dissertation, Harvard University, Cambridge, Mass., January 1968.

42. R. Narasimhan, "Labeling schemata and syntactic descriptions of pictures," Information and Control, vol. 7, pp. 151-179, 1964.

43. R. Narasimhan, "Syntax-directed interpretation of classes of pictures," Comm. ACM, vol. 9, pp. 166 - 173, 1966.

44. B. H. McCormick, "The Illinois pattern recognition computer (ILLIAC III)," Digital Computer Lab., University of Illinois, Urbana, Illinois, Rept. 148, 1963.

45. R. Narasimhan and B. H. Mayoh, "The structure of a program for scanning bubble chamber negatives," Digital Computer Lab., University of Illinois, Urbana, Ill., File 507, 1963.

46. J. L. Pfaltz and A. Rosenfeld, "Sequential operations in digital picture processing," J. ACM, vol. 13, pp. 471-494, 1966.

47. T. V. Griffiths and S. R. Petrick, "On the relative efficiencies of context-free grammar recognizers," Comm. ACM, vol. 8, pp. 289-300, May 1965.

48. R. W. Floyd, "The syntax of programming languages--a survey," IEEE Trans. Electronic Computers, vol. EC-13, pp. 346-353, August 1964.

49. A. C. Shaw, "A formal picture description scheme as a basis for picture processing systems," Information and Control, vol. 14, pp. 9-52, 1969.

50. J. L. Pfaltz and A. Rosenfeld, "Web grammars," Proc. Joint International Conference on Artificial Intelligence, Washington, D. C., 1969.

51. G. U. Montanari, "Separable graphs, planar graphs, and web grammars," Computer Science Center, University of Maryland, College Park, Md., Tech. Rept. 69-96, August 1969.

52. N. Wirth and H. Weber, "Euler: A generalization of Algol, and its formal definition," Comm. ACM, vol. 9, pp. 13-25, January 1966.

53. T. G. Evans, "A grammar-controlled pattern analyzer," Proc. IFIP Congress 1968, vol. 2, pp. 1592-1598, August 1968.

54. E. M. Gold, "Language identification in the limit," Information and Control, vol. 10, pp. 447-474, 1967.

55. R. Sherman and G. W. Ernst, "Learning patterns in terms of other patterns," Pattern Recognition, vol. 1, pp. 301-313, July 1969.

56. A. Rosenfeld, H. K. Huang, and V. B. Schneider, "An application of cluster detection to text and picture processing," IEEE Trans. Information Theory, vol. IT-15, pp. 672-681, November 1969.

57. R. W. Sauvain and L. Uhr, "A teachable pattern describing and recognizing program," Pattern Recognition, vol. 1, pp. 219-232, March 1969.

58. T. G. Evans, "Grammatical inference techniques in pattern analysis," presented at the Third International Symposium on Computer and Information Science (COINS-69), Bal Harbour, Fla., December 1969.

59. J. J. Horning, "A study of grammatical inference," Computer Science Dept., Stanford University, Stanford, Calif., Tech. Rept. CS 139, August 1969.

60. U. Grenander, "Foundations of pattern analysis," Div. of Applied Math., Brown University, Providence, R. I., Tech. Rept. 2 (Project Graphics), 1967.

61. U. Grenander, "Syntax-controlled probabilities," Div. of Applied Math., Brown University, Providence, R. I., 1967.

62. T. E. Harris, The Theory of Branching Processes. Berlin: Springer-Verlag, 1963.

63. T. L. Booth, "Probabilistic representation of formal languages," IEEE Conference Record Tenth Annual Symposium on Switching and Automata Theory, pp. 74-81, October 1969.

64. M. M. Kherts, "Entropy of languages generated by automated or context-free grammars with a single-valued deduction," Naucho-Tekhnicheskaia Informatsia, ser. 2, no. 1, pp. 29-34, Jan. 1968.

65. K. S. Fu and T. J. Li, "On stochastic automata and languages," to appear in the International Journal on Information Sciences.

66. J. E. Hopcroft and J. D. Ullman, Formal Languages and their Relation to Automata. Reading, Mass.: Addison-Wesley, 1969.

67. A. Gill, Introduction to the Theory of Finite-State Machines. New York: McGraw-Hill, 1962.

68. C. A. Ellis, "Probabilistic languages and automata," Dept. of Computer Sciences, University of Illinois, Urbana, Ill., Rept. 355, October 1969.

69. A. Salomaa, "Probabilistic and weighted grammars," Information and Control, vol. 15, pp. 529-544, 1969.

70. T. Huang and K. S. Fu, "On stochastic context-free languages," School of Elec. Engrg., Purdue University, Lafayette, Ind., Tech. Rept. TR-EE 70-11, March 1970.

71. M. O. Rabin, "Probabilistic automata," Information and Control, vol. 6, pp. 230-245, 1963.

72. M. A. Harrison, Introduction to Switching and Automata Theory. New York: McGraw-Hill, 1965.

73. T. L. Booth, Sequential Machines and Automata Theory. New York: Wiley, 1967.

74. S. H. Caldwell, Switching Circuits and Logical Design. New York: Wiley, 1958.

75. D. J. Rosenkrantz, "Programmed grammars--a new device for generating formal languages," Ph.D. dissertation, Columbia University, New York, 1967.

76. D. J. Rosenkrantz, "Programmed grammars, a new device for generating formal languages," IEEE Conference Record Eighth Annual Symposium on Switching and Automata, pp. 14-20, October 1967.

77. B. A. Chartres and J. J. Florentin, "A universal syntax-directed top-down analyzer," J. ACM, vol. 15, pp. 447-464, 1968.

78. N. Chomsky and M. P. Schutzenberger, "The algebraic theory of context-free languages," in Computer Programming and Formal Systems. P. Braffort and D. Hirschberg, Eds., North-Holland, Amsterdam, pp. 118-161, 1963.

UNCLASSIFIED
Security Classification

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)
School of Electrical Engineering
Purdue University

2a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

2b. GROUP

3. REPORT TITLE
NONPARAMETRIC AND LINGUISTIC APPROACHES TO PATTERN RECOGNITION

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)
Scientific Report

5. AUTHOR(S) (Last name, first name, initial)
'wain, Philip H.
Fu, King Sun

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
June 1970	184 + 8	78

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
AFOSR Grant 69-1776	TR-EE 70-20
b. PROJECT NO.	
c.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
d.	

10. AVAILABILITY/LIMITATION NOTICES
UNLIMITED

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Air Force Office of Scientific Research

13. ABSTRACT

This report investigates two approaches to pattern recognition which utilize information about pattern organization. First, a nonparametric method is developed for estimating the probability density functions associated with the pattern classes. The dispersion of the patterns in the feature space is used in attempting to optimize the estimate. The second approach involves the structural relationships of pattern components, an approach called "linguistic" because it employs the concepts and methods of formal linguistics.

The nonparametric density estimation technique is shown to produce acceptable results with real data and demonstrates a definite advantage over a parametric procedure when multimodal data is involved.

Two alternative techniques are investigated for analyzing linguistic descriptions of patterns. Stochastic automata are considered as recognizers of stochastic pattern languages. The other technique is a stochastic generalization of the recently proposed programmed grammar which is developed as a grammar for pattern description.

DD FORM 1 JAN 64 1473

UNCLASSIFIED
Security Classification