A Classifier Design Technique for Discrete Variable Pattern Recognition Problems

JAMES C. STOFFEL, MEMBER, IEEE

Abstract—**This paper presents a new computerized technique to aid the designers of pattern classifiers when the measurement variables are discrete and the values form a simple nominal scale (no inherent metric). A theory of "prime events" which applies to patterns with measurements of this type is presented. A procedure for applying the theory of "prime events" and an analysis of the "prime event estimates" is given. To manifest additional characteristics of this technique, an example optical character recognition (OCR) application is discussed.**

Index Terms—**Classification, discrete variables, interactive pattern recognition, nonparametric pattern recognition, n-tuple selection, optical character recognition (OCR), pattern recognition, prime events.**

I. INTRODUCTION

THE DESIGN of automatic classification devices often relies on one or a number of mathematical procedures which manifest the statistical structure of samples from the classes to be discriminated. Such mathematical procedures have been effectively incorporated in interactive computer environments (see Kanal [1]). There exist few mathematical techniques, however, to deal with patterns whose measurement variables have discrete, type-II values. The following will describe the unique characteristics of discrete type-II variables, the difficulties in dealing with these variables in pattern-recognition problems, and a new theoretical and practical procedure to handle such variables.

II. TYPES OF VARIABLES

In a classical manner, a pattern will be represented hereafter as a vector, $v = [v_1 v_2 \cdots v_d]$. The elements v_i in the vector represent the values of specific measurements, and the dimension of the vector d corresponds to the number of measurements used to represent the pattern. The measurement variables will be divided into three categories. Following the definitions of Sammon [2], the categories will be labeled continuous, discrete, type-I, and discrete, type-II.

The continuous category is composed of variables which may take on any value in a prescribed range of the real line.

The second type of variable is defined as discrete, type-I. This group will include those variables which take on only discrete values, but which represent samples from some underlying continuum. An example of these discrete, type-I variables is age. Although it is typically given as integer years, the measurement value is a sample from a quantized continuum, time since birth.

A variable in the third category, called discrete, type-II, may take on discrete values, but there is no apparently significant ranking of these values. An example of such a variable is the blood type of a sample. The possible values are: A, AB, O, etc. These values are discrete and represent a nominal, rather than an ordinal scale.

For cases when the measurement variables are discrete, type-II, significant practical difficulties may arise in the design of accurate classifiers. It is characteristic of the discrete, type-II variables that a change of a single variable value, from one level to another, may represent a significant change in the measured item. For example, two specimens which have identical measurement values, with the exception of blood type, may have significantly different responses to certain pathogens.

Unlike the continuous category there is no incremental variation in the discrete, type-II variable values which will leave the pattern effectively unaltered. Due to this factor and the lack of an inherent ordering principle for the variable values, it is difficult to describe the probability distribution of the measurement vectors by a simple parametric model. If a generally applicable model existed with relatively few parameters, the designer might confidently estimate the parameters of the distribution and then approximate an optimal discriminate rule such as Bayes' (Kanal [3]).

As an example, consider the case when the measurement vector has a finite dimension and each variable takes on a finite number of values. From a statistical point of view, it would appear useful to estimate the probabilities of *each* of the possible measurement vectors. For this "multinomial model" (Foley [4]), assume that all measurements are binary valued and that the measurement vector has dimension ten. These conditions imply that there are 1024 parameters, probabilities for the measurement vectors, to estimate. Since a number of these parameters will be less than one one-thousandth, a large number of samples of each class must be obtained to guarantee confidence in these estimates. Foley [4] implies that 3072 to 4096 samples per class should be utilized to yield confidence in the classifier design. Such large sample sets are often unobtainable, however.

Manuscript received March 6, 1973; revised September 24, 1973. This work was supported in part by a research grant to Syracuse University from the Rome Air Development Center, Rome, N. Y.

The author is with the Research Laboratories Department, Xerox Corporation, Rochester, N. Y. 14644.

Augmenting the statistical tools which aid the practical classifier designer are procedures which are based on "geometric principles." Rather than estimate statistical parameters, a designer may begin to treat the set of possible measurement vectors as members of a vector space, wherein he seeks a relatively simple discriminate rule to be specified in terms of a metric for that space. Exemplars of this approach are "nearest neighbors" classification rules (Fix, Hodges [5]), and Fisher's Linear Discriminant (Fisher [6], Sammon [7]).

For measurement vectors with discrete, type-II variables, however, the Euclidean metric is not generally applicable as a similarity measure (Hills [8]). Furthermore, the arbitrary nature of the nominal scale makes alternate metrics difficult to derive (Lance, Williams [9]). The geometric principles which lead to useful design tools for classification tasks with continuous variable measurement vectors are not generally applicable to the cases with discrete, type-II variables.

III. APPROACHES TO HANDLING DISCRETE VARIABLES

This paper attempts to present a general "tool" to assist the classifier designer in extracting classificatory information from data with discrete, type-II variables. This technique, like all other in pattern recognition, will not solve every classification problem; however, it will add to the designer's ability to handle discrete variable data. The initiative for the work reported here was the work done in the development of a "discrete variable subsystem" for OLPARS, by Sammon [10]. It is the author's belief that the analytical techniques developed in this paper are best suited to an interactive environment such as described by Kanal [1].

Various approaches have been taken to deal with the difficulties of discrete, type-II variables as cited above. Linhart [11] and Cochran and Hopkins [12] were among the first to publish the inherent difficulties in handling classification problems with discrete, type-II variables. A number of others have utilized a variety of techniques in an attempt to overcome the difficulties with variables of this category.

Viewing the problem statistically, a number of restrictions have been imposed on the general multinomial model of the probability distribution of the measurement vectors. If one assumes that the variables are mutually independent, then there are only d parameters to estimate for measurement vectors of dimension d with binary-valued variables. Regarding this as too restrictive, one might assume the "latent class" model of Lazarsfeld [13] to represent a class of measurement vectors. This model would represent the class as a summation of subclasses, each of which has variables which are mutually independent. If there were s subclasses, then one would have to estimate $s \times (d + 1)$ parameters for measurement vectors of dimension d whose measurements were binary valued. Investigations of the parameter estimation of this model have been carried out, by Anderson [14] and McHugh [15]; however, this is not a generally applicable model.

Another restricted model which has been proposed is the "logit" model. For this model, the logarithm of the probability distribution of the measurement vectors is represented as follows:

$$\log \{P(v_1,v_2,\cdots,v_d)\} = a_0 + \sum_{i=1}^{d} (-1)^{v_i} a_i \qquad (1)$$

where $v_i \in \{0,1\}$ and a_i are constants. Schaefer [16] utilized this model for binary-valued measurement variables, which are by definition discrete, type-II. (A generalization of this model and additional details may be found in Good [17], Gilbert [18], and Cox [19].) This model, as well as the restricted models given above, represents assumptions about the classes being dealt with. For practical applications these various assumptions must be justified before the models become of value.

An alternate technique which has been employed is to develop a finite series expansion for the general multinomial probability distribution. By a judicious choice of the basis set, a concise description of the total probability distribution may be possible if problem information constrains some coefficients to zero. Examples of this approach are Bahadur [20], Abend *et al.* [21], and Chow [22], [23]. This approach has yielded limited results.

A variety of "geometric" techniques have also been applied to the discrete, type-II variable classifier design task. The "near neighbor" concept of Fix and Hodges [5] was applied unsuccessfully by Hills [8] to discriminate discrete, type-II data. Gilbert [18] examined the applicability of Fisher's *linear discriminant* to the classification of a restricted class of measurement vectors with binary variables. Furthermore, numerous *similarity metrics* have been proposed and applied to classification tasks. Among these are the "matching metric" of Sokal and Sneath [24], the "hierarchical" clustering metrics of Lance and Williams [9], and a "probabilistic" metric defined by Goodall [25].

Still another alternative for dealing with discrete, type-II variables is to transform them into continuous variables. Examples of such transformations may be found in Sammon [2].

The above techniques have not been valueless, but there remains a need for a general approach to extracting classificatory information from the sample data. An approach which makes few *a priori* decisions about the data is "measurement selection." This technique attempts to eliminate from the measurement vector variables which contribute little or nothing to the classification accuracy. Hills [8], for example, selects the "best" subset of measurement variable by estimating the information divergence, Fortier and Solomon [26] incorporate the product moment correlations of the variables in a figure-of-merit selection algorithm, and McQuitty [27] applies a clustering algorithm. These procedures tend to be ineffective or

to require exhaustive, impractical computation for discrete, type-II variable data. Some theoretical justification for the lack of positive results in this area may be found in Elashoff *et al.* [28] and Toussaint [29].

IV. PRIME EVENTS

A new theoretical point of view is presented in this section which deals with discrete, type-II variable measurement vectors. The theory is generally applicable to this type of data; i.e., no *a priori* decisions about the form of the underlying probability distributions are required.

A pattern is available to a classifier designer in the form of a d-dimensional vector, $\boldsymbol{v} = (v_1, v_2, \cdots, v_d)$, the measurement vector. The set of possible measurement vectors is thus comprised of vectors whose elements lie in the range of the measurement functions associated with each element in the vector. The set of possible measurement vectors forms a set of *events* in a sample space.

An event in the sample space described above is completely specified by a d-dimensional vector. The term, event, will be utilized hereafter to denote a measurement vector in which a subset of the d elements have specified values. The remaining elements are unspecified and will be represented by a "—" in the vector. Hence, event $e_1 = (1,2,5)$ represents the state that measurements one, two, and three take on values 1, 2, and 5, respectively. Furthermore, event $e_2 = (1,2,—)$ is defined to represent the state that measurements one and two take on values 1 and 2, respectively. Measurement three is *not specified.*

One of the significant characteristics of an event is its ability to "**cover**" other events. An event e_1 will be said to cover an event e_2 if and only if every element of e_1 which has a specified value equals the value of the corresponding item in e_2. Furthermore, if an element in e_1 is not specified, then any value or a "not specified" value may exist for the corresponding element of e_2.

To exemplify the covering principle, examine the following events:

$$e_1 = (2,—,—)$$

$$e_2 = (2,1,0)$$

$$e_3 = (2,2,—).$$

Using the above definition, one may see that event e_1 covers e_2 and e_3. However, e_2 does not cover e_1 or e_3, and e_3 does not cover e_1 or e_2.

Due to the covering capacity of an event, one may note that an event partitions the set of possible measurement vectors into two subsets. One subset comprises the measurement vectors which are covered by the event, and the other subset contains all measurement vectors not covered by the event. The above principle is easily incorporated in the design of a classifier. For each class, a collection of events which will henceforth be termed the *definition set* is assigned. The classification rule is, then, to assign a measurement vector to the class whose definition set contained an event which covered the measurement vector. If the measurement vector is not covered by a member of any definition set, then it is rejected.

To implement the simple classification procedure described above, a number of major questions must be answered. First of all, what events should theoretically be contained within the definition sets of each class? The number of possible events is typically quite large. For example, a ten-dimensional measurement vector with binary-valued elements will have 3^{10} events which must be considered in making up a definition set. Further, the sets of events may overlap one another in varying degrees, and various percentages of the one or more classes may be covered by one event. Hence, there are potentially complex tradeoffs in the error rates for various sets of events used as a definition set.

Second, for a specific classification problem, how does one generate the theoretically ideal events as defined by the answer to question one?

Finally, how does sample information from a practical design problem become incorporated in the design of definition sets for each class?

The remainder of this section will be devoted to answering the first question; the next section will deal with questions two and three. Therefore, the immediate goal is to ascertain just what events should be contained within the definition set of a particular class.

Two types of errors are of concern when using events to discriminate between classes. Type-I errors will be defined as the occurrence of a measurement vector from class C_j which is covered by an event in the definition set of class C_i, for $i \neq j$. Type-II errors will be defined as the occurrence of a measurement vector from class C_j which is not covered by an event in the definition set for class C_j. It is not always possible to reduce the occurrence of errors to zero, since classes may overlap. The optimality criterion which is utilized, therefore, is the minimization of the *average probability of error.*

Another criterion is placed on the theoretically optimal solution. This second criterion states that the set of events which comprise the definition set for each class must be minimal in number. Among other things, this criterion introduces the practical consideration of storage for a classification algorithm.

The first criterion for the optimal definition set may be met by Bayes' rule, Kanal [3]. For the case under consideration, the rule states that the minimum average probability of error is obtained when the classifier assigns a measurement vector to the class which yields the largest *a posteriori* probability. Using the notation $p(\boldsymbol{v} \mid C_j)$ for the conditional probability of measurement vector $\boldsymbol{v}$, given class C_j as a source and $p(C_j)$ as the probability of occurrence of a vector from class C_j, then the optimal rule assigns to class C_j a sample $\boldsymbol{v}$ whenever $p(\boldsymbol{v} \mid C_j)p(C_j) \geq p(\boldsymbol{v} \mid C_i)p(C_i)$ for all i.

From the above, one may deduce that a definition set comprised of a measurement vectors which the above rule has assigned to a specific class would satisfy criterion one. However, criterion two may not be satisfied by the col-

lection of measurement vectors. Hence, the optimal set of events is not immediately obtained.

A particular set of events will be shown to be a sufficient collection from which an optimal definition set may be selected. The members of this set are labeled "**prime events.**" A prime event is defined as an event which *covers* only those measurement vectors which a Bayes' Rule would assign to *one class*; furthermore, a prime event may not be covered by another prime event.

The fact that the optimal definition sets may be composed of events from the set of prime events may be easily proved by contradiction. Under the assumption that the optimal design set includes a nonprime event e_j, it follows that e_j must either be covered by a prime event e_p, and thus e_p may replace e_j in the design set, or e_j covers measurement vectors assigned by Bayes' Rule to more than one class. The second alternative is not possible, since the optimal definition set will not contradict Bayes' Rule; therefore, the prime events form a sufficient set.

In order to clarify the ideas expressed above, an example will be given utilizing the set of three-dimensional binary-valued measurement vectors. The example is that of a two-class problem plus reject class, and the class conditional probabilities for the measurement vectors are given below. The example shows the assignment of the measurement vectors and the prime events associated with the problem. The optimal design sets for each class are then listed.

Example 1

Measurement vectors	$p(v_i \mid C_1)$	$p(v_i \mid C_2)$	$p(v_i \mid R)$
000	0	0	0.25
001	0	0	0.25
010	0.4	0.2	0
011	0.4	0.2	0
100	0.1	0.3	0
101	0.1	0.2	0
110	0	0.1	0
111	0	0	0.50

Assume that the probability of each class is 0.45; namely, $p(C_1) = p(C_2) = 0.45$ and $p(R) = 0.10$. The decision rule described above may be rewritten as follows.

Assign v_i to class C_j whenever

$$p(v_i \mid C_j)p(C_j) \geq p(v_i \mid C_k)p(C_k)$$

for $C_j, C_k \in \{C_i, C_2, R\}$.

Utilizing this rule, the measurement vectors will be assigned in the following fashion:

$$\text{Class } C_1 = \{(0,1,0),\ (0,1,1)\}$$

$$\text{Class } C_2 = \{(1,0,0),\ (1,0,1),\ (1,1,0)\}$$

$$\text{Reject} = \{(0,0,0),\ (0,0,1),\ (1,1,1)\};$$

the prime events for class C_1: $\{(0,1-)\}$, for class C_2: $\{(1,0,—),\ (1,—,0)\}$; the chosen optimal design sets are:

$$E_1 = \{(0,1,—)\}$$

$$E_2 = \{(1,0,—),\ (1,—,0)\}.$$

Example 1 is intentionally simple and does not reflect some of the more complex characteristics of prime events and definition sets. There is a potentially large number of prime events which may be associated with a particular classification problem. The number of these prime events may be as large as the number of measurement vectors which they cover. The magnitude of their number and their characteristics are, furthermore, problem-dependent facts.

The potential variability in the number of prime events and their complex covering properties results in definition sets of various sizes and (problem-dependent) characteristics. In addition, the definition sets for a particular classification problem are not unique, even though they may be composed of only prime events.

The suggested procedure for designing a classifier requires that one first generate the prime events for the specific problem at hand. The next step is to select a minimal size set of prime events to make up the definition sets. These procedures are discussed in the next section.

V. GENERATION OF EVENTS

The next question to be dealt with is the method of creating the optimal design set for a specific class. In what follows, a procedure will be given which is capable of creating all of the prime events associated with a specified set of measurement vectors. A procedure for selecting the final design set will then be described.

It is intended that the generation procedure be performed for each class to yield the optimal set of events for that class. Thus, for a six-class problem six sets of events will be generated. The reject class then becomes the remainder of the set of measurement vectors which are not covered by any of the six sets of events.

As throughout the discussion of prime events, the procedure described below will make use of the knowledge of the class conditional probabilities of the measurement vectors, along with the probabilities of occurrence of the classes. These quantities must be estimated in most practical cases.

The first step in the generation procedure is to form two lists of measurement vectors, L and NL. List L is comprised of all the measurement vectors which the Bayes' decision rule would assign to the chosen class. List NL is comprised of all the measurement vectors which are not assigned to the chosen class.

The procedure of generating prime events uses the list L to create new events and forms a new list, LP, of potential prime events. The creation of new lists continues until no new events are generated.

New events are created from list L by combining a pair of events in the list. The combining process for two events in L is performed using the lattice binary operation "join" for each variable. To make clear this procedure, the operation of combining variables will first be dealt with.

The result of combining variables may be expressed as a binary operation with a simple rule. If the two variable values differ, then the result is a "don't care" value. If

the variables are the same, the resultant variable has that same value. When combining a "don't care" value with anything, the result is a "don't care" value. For ternary-valued variables, the table below will demonstrate the combination rules described above:

v'	v''	$(v' + v'')$
0	0	0
0	1	—
0	2	—
0	—	—
1	0	—
1	1	1
1	2	—
1	—	—
2	0	—
2	1	—
2	2	2
2	—	—
—	0	—
—	1	—
—	2	—
—	—	—

where "—" signifies a "don't care."

Now, to combine two events in L which have ternary-valued variables, the above operation will be applied "pointwise" to each variable. The combination of the following two events will help clarify this concept:

$$(2,—,1) + (2,1,0) = (2,—,—).$$

In this way, any two events derived from the same measurement space may be combined to form a new event. The process of combining events will henceforth be termed "merging."

The procedure for generating the set of prime events begins by merging all pairs of events in the list L. Each new event, the result of merging two events in L, is checked to see if it covers any members of NL. If it does cover a member of NL, then the event will not be a prime event nor will any event which covers this new event. This follows from the restriction that a prime event may not cover measurement vectors which a Bayes' decision strategy would assign to two different classes. The new event is then disregarded.

On the other hand, if the result of merging two events in L does not cover any member of NL, then this event will be a potential prime event, and it is stored in list LP. All new events generated from L are checked and only those new events which are truly *potential prime events* are stored in LP. Also added to LP is any event in L which did not yield a *potential prime* event when it was merged with any of the other events in L.

The procedure forms all pairwise mergers in list L and adds to list LP those new events which only cover measurement vectors assigned to a specific class. The list LP will also include those events from L which did not yield a potential prime event upon merging with any of the events in L.

The list LP is next treated as list L and the merging process is repeated as before. Each pair of events in LP is merged and a check is performed to see if the new event is a potential prime event.

This formation of new lists continues until list L equals list LP. This occurs when no merger of a pair of events in L yields an event which is a potential prime event and is different from each member of L.

An example of the generation procedure will be given below to help clarify the specific details of the algorithm. The example problem is a two-class problem with three-dimensional ternary-valued variables. The procedure will be applied to derive the set of prime events associated with one of the classes, C_1.

Let the set of measurement vectors which a Bayes' decision rule would assign to class C_1 be as follows: $\{(2,0,0), (2,1,0), (2,2,0), (2,0,1)\}$. The remaining measurement vectors will be either in class C_2 or a reject class. The algorithm begins by forming list L.

List L

1: (2,0,0)
2: (2,1,0)
3: (2,2,0)
4: (2,0,1)

Indices have been assigned to each measurement vector so that they may be referenced more easily, i.e., $e_1 \triangleq (2,0,0)$.

Next, one merges events e_1 and e_2:

$$(2,0,0) + (2,1,0) = (2,—,0) \triangleq e_{12}.$$

This event is checked against the list NL with the result that no member of that list is covered by event e_{12}. Thus, e_{12} is placed in list LP.

The events e_1 and e_3 are then merged:

$$(2,0,0) + (2,2,0) = (2,—,0) = e_{13}.$$

Since the event e_{13} is already in LP, no further processing is required.

The remaining combinations yield:

$$e_{14} \triangleq (2,0,—)$$

$$e_{23} \triangleq (2,—,0)$$

$$e_{24} \triangleq (2,—,—)$$

$$e_{34} \triangleq (2,—,—).$$

The events e_{14}, e_{24}, e_{34} are not retained, since they cover members of NL. Event e_{23} is already stored on LP and hence is disregarded.

Event e_4 is next added to list LP since it merged with no member of L to yield a potential prime event.

$$LP = \{(2,—,0), (2,0,1)\}.$$

Checking LP and L, one will find them different. Therefore, LP becomes list L and a new empty list is introduced as LP:

List L	List LP
$e_{12}(2,—,0)$	(Null).
$e_4(2,0,1)$	

Repeating the merging process:

$$(2,—,0) + (2,0,1) = (2,—,—) \triangleq e_{124}.$$

Event e_{124} covers elements of list NL and therefore is disregarded.

No further mergers are possible in list L, and thus e_{12} and e_4 are added to LP:

$$LP = \{2,—,0), (2,0,1)\}.$$

Checking LP and L, it will be seen that they are identical, and thus the procedure *stops*. The set of prime events for class C_1 is in both lists, L and LP.

It can be shown that the above procedure will generate all, and only, the prime events for a specific classification problem. To manifest that the procedure will generate *all* of the prime events, this fact will be negated and a contradiction will be shown to follow. Pursuing this method, let there exist some prime event e_p which is not generated by the procedure described above. Event e_p covers a set of measurement vectors which by Bayes' rule will be classified as belonging to one class. Thus a pair or any subset of these vectors, when merged, will yield an event which is covered by e_p and hence covers no member of another class. The generative procedure described above discards only those mergers which cover measurement vectors of two or more classes. Hence, the mergers of measurement vectors which are covered by e_p will be retained. Successive, exhaustive, pairwise merging of the retained events which are covered by e_p will eventually generate event e_p, as it represents the merger of all measurement vectors which it covers. But this contradicts the assumption that there exists a prime event which is not generated by the above procedure.

To show that *only* prime events are generated via the above procedure, a proof by contradiction will also suffice. Assume that an event e_n is generated but that e_n is not a prime event. There are two restrictions of prime events which e_n may have violated. First of all, e_n may cover measurement vectors from two or more classes. This may be ruled out since the above generative procedure does not retain events as potential prime events for a specific class if that event also covers measurement vectors of another class, members of NL. The second prime event restriction is that a prime event may not be covered by another prime event. Event e_n may not be covered by some prime event e_p, however, since the generative procedure would attempt to merge e_n and e_p and yield event e_p, which would be retained. Event e_n would not remain with the final list of prime events which was shown above to be complete. Therefore, the assumption that event e_n exists is false, and the generative procedure has been shown to yield only prime events.

The final theoretical step in the creation of the optimal definition sets for the classes of a specific problem is the selection of the *minimum* size set of the generated prime events. The measurement vectors which a definition set must cover are the same vectors which were placed in list L to generate the prime events. The goal is, therefore, to select a minimal-size cover for these measurement vectors.

One method of selecting the minimal-size covering is to exhaustively examine the power set of the set of prime events. One could then select from this set the smallest set of prime events which covered the required measurement vectors. Alternate procedures for selecting this minimal-size cover may be found in the procedures developed to "minimize Boolean functions," Prather [30]. For a theoretical analysis, the existence of some procedure is sufficient. Thus, the problem of selecting a minimal cover will be considered answered until the practical application of this theory is discussed in the next section.

The Prime-Event-Generation (PEG) algorithm will operate to minimize a function of the discrete variables. Algorithms which minimize the "cost" of a Boolean function, e.g., Prather [30], yield similar results. Both procedures generate covers for a set of discrete points. However, the PEG algorithm will operate on other than binary-valued variables, with various mixes of multivalued variables, and with more than the three classes of points, "trues," "falses," and "don't cares," which define Boolean functions.

This section has provided theoretical answers to the questions of what events should be utilized in a classifier and how these events may be generated and selected. The next section will deal with the practical classifier design task and the application of the theory of prime events to it.

VI. APPLICATION OF PRIME EVENT THEORY

The following is the proposed method for dealing with design problems wherein the probabilities of the measurement vectors are not known. When one is not aware of the class conditional probabilities of the measurement vectors, optimal design procedures may only be estimated. The set of prime events will serve as a goal for the design methods described below. However, only sample information will be utilized in the procedure.

The proposed method of dealing with sample information is to form estimates of the true prime events for the particular classes. The mechanism for creating these estimates is to incorporate samples of the classes as the assigned measurement vectors were incorporated in the prime-event-generation algorithm described in the previous section. To obtain the prime events associated with a class C, one places the samples from class C in list L and the samples from all other classes in list NL. The prime-event-generation algorithm then proceeds, as described above, to merge the samples in list L and to add potential prime events (truly estimates) to list LP. Proceeding as previously described, the generation algorithm will halt when list L equals list LP.

The task of extracting the minimal size cover as a definition set is also encumbered by the practicality of dealing only with estimates of prime events. For practical classifiers, it is economically advantageous to store as few prime event estimates as possible. However, the selection of the smallest set which simultaneously covers the set of samples of that specific class is not guaranteed to yield the best classifier accuracy; it may, in fact, degrade the performance over that of a larger set. Thus, the selection of the definition set may incorporate a tradeoff in the number of prime event estimates and their ability to cover all, and only, the measurement vectors which Bayes' rule would assign to that class. When the number of prime event estimates is large, a cover-selection algorithm such as given by Prather [30] may be incorporated. If but a few estimates are generated, it may be practical to include all of them in the definition set. This decision is often dependent on problem constraints which are specific to the design tasks; hence, no rule is given here. An example-selection procedure is described in the design application presented in Section VII.

Two new facilities are added to the fundamental prime-event-generation procedure to increase its flexibility when dealing with sample information. The first facility is a threshold, θ_A, which is utilized when checking if an event covers a member of list NL. An event will be treated as though it did not cover a member of NL if the event truly covered less than or equal to θ_A percent of list NL.

The normal setting of θ_A is 0. It may be raised to other percentage values, however. This flexibility may prove helpful when a measurement vector occurs as a sample in two different classes. If sufficient numbers of samples are available to confidently estimate the Bayes' rule for classifying such measurement vectors, they may simply be removed from classes other than the properly assigned class. The generation procedure may then proceed as before. However, if insufficient confidence is established with the available samples, then threshold θ_A may be raised above 0.

The second facility which is added to the fundamental generation algorithm is a threshold, θ_B. An event will be considered to cover a measurement vector in list NL if the number of specified variables in the event which differ from those of the measurement vector is less than θ_B.

To exemplify the use of θ_B, assume that vector (2,1,0,1,0) is in list NL. If θ_B is set at the normal value 1, then the event (2,—,1,—,0) will be considered, as described in the previous section, not to cover the measurement vector. However, if θ_B is raised to 2, then the event will be treated as though it covers the measurement vector, since only one specified variable in the event differs in value from that of the measurement vector.

Both θ_A and θ_B are utilized to compensate for the finite sample sizes in practical problems. By raising θ_A above 0, one is not forced to make decisions based on a small number of samples in a very large measurement space. The variable θ_B is utilized to provide greater confidence that a generated event will not cover samples of a class represented in list NL. An increase in θ_B from 1 to 2 will compensate for the absence of a sample in NL which is different in only one variable value from a sample presently in NL. In this way one may compensate for some deficiencies in the samples on list NL.

The method of incorporating samples in the basic prime-event-generation algorithm provides a mechanism for extracting the classificatory information from the discrete, type-II variable data. This algorithm with the added flexibilities provided by θ_A and θ_B is defined as the Prime-Event-Generation algorithm (PEG) and it represents the fundamental analytic mechanism which is presented as a classifier design tool.

The important qualities of the prime-event-estimation procedures are noted below. An example application of these procedures is discussed in the section which follows.

The first notable characteristic of the prime-event-estimation procedure is its generality. The technique makes no *a priori* decisions about the form of the underlying probability distributions for the measurement vectors. The PEG algorithm is a tool for extracting from the samples of the classes that "structural" information which will make possible the discrimination of these classes, regardless of the underlying probability distributions associated with each class.

A second characteristic of the PEG procedure which should be noted concerns the statistical confidence of estimated parameters. As noted earlier, for discrete, type-II variable classifier design tasks, the number of possible measurement vectors is often too large in number to obtain confident estimates of their probabilities of occurrence using practical sample sizes. The merging process of the PEG algorithm yields events which may have a number of variables not specified in value. These events may truly be considered as members of the subspace defined by the variables which are specified. Hence, there will be less possible variation in that subspace and greater confidence may be placed in an estimate of the probability of the event occurring than in the estimate of measurement vectors.

To exemplify the above concept, assume there exists a two-class recognition problem with ten-dimensional, binary-valued measurement vectors. If one treats the vectors as a positional binary number, then the measurement vectors may be denoted by a decimal number in the range from 0 to 1024. Using this notation, let the samples of class C_1 be the set $\{0,1,2,\cdots,255\}$, and let the samples of class C_2 be the set $\{256,257,\cdots,767\}$ (see Fig. 1).

For the example described above, one may estimate the probabilities for each of the possible measurement vectors. However, there is at most one sample of any of the possible measurement vectors. Also, the greatest difference in the number of samples of a specific measurement vector for the two classes is one. Furthermore, there are only 768 samples with which to estimate the 2048 class-conditional probabilities of the measurement vectors. Little statistical confidence may be placed in these estimates.

On the other hand, one may note that all samples in class C have the first two measurements equal to 0. Furthermore, class C_2 samples have measurements one and two equal to the two-tuple (0,1) or (1,0). A discriminatory event for class C_1 could thus be (0,0,—,—,—,—,—,—,—,—) and a pair for class C_2 could be (0,1,—,—,—,—,—,—,—,—) and (1,0,—,—,—,—,—,—,—,—). All of these events cover 256 samples each, and greater confidence may be placed in estimating an accurate classification rule for these events than for the measurement vectors considered above.

A third point to note about the PEG procedure is that it extracts from the samples the within-class similarities which simultaneously discriminate the specific classes from one another. The prime-event estimates simultaneously fit a specific class and discriminate this class from the other classes. Classifier design tools which simultaneously perform these two functions are rare. Often a procedure relies strictly on its facility to discriminate the samples, e.g., Ho-Kashyap [31]. The application of such a procedure requires care (see Foley [4] and Cover [32]). Furthermore, the possibility of a reject class is annulled.

On the other hand, there are procedures which concentrate on fitting each class with some description; then, classification becomes a matter of determining the best description for a sample from the set of class descriptions. However, a useful description for classification will emphasize the features which are different in separate classes, and not merely describe a class in some least mean-square error sense (see Fukanga [33]). The prime-event-generation procedure provides fitting information, but by definition the events have the ability to discriminate the classes.

Another major point which should be emphasized is the ability of the PEG algorithm to detect high-order statistical structure without requiring exhaustive searches. For many practical classification problems, the dimension of the measurement vector is such that only first- and second-order statistics are examined. Higher order statistical analysis would be too time consuming. The example below will help manifest the ability of the PEG procedure to enable higher order statistical analysis.

Assume that a two-class discrimination problem had fifty-dimensional, binary-valued measurement vectors. Let the probabilities for measurements ten, twenty, and thirty, when treated as a three-tuple, be:

$v' = (v_{10}\ v_{20}\ v_{30})$	$P(v' \mid C_1)$	$P(v' \mid C_2)$
0 0 0	$\frac{1}{4}$	0
0 0 1	0	$\frac{1}{4}$
0 1 0	0	$\frac{1}{4}$
0 1 1	$\frac{1}{4}$	0
1 0 0	0	$\frac{1}{4}$
1 0 1	$\frac{1}{4}$	0
1 1 0	$\frac{1}{4}$	0
1 1 1	0	$\frac{1}{4}$

Finally, assume that all other measurements have a fifty percent probability of being a 1 for both classes.

Class C_1 Samples

v_0 : (0,0,0,0,0,0,0,0,0,0)
v_1 : (0,0,0,0,0,0,0,0,0,1)
v_2 : (0,0,0,0,0,0,0,0,1,0)
.
.
.
v_{255}: (0,0,1,1,1,1,1,1,1,1)

Class C_2 Samples

v_{256}: (0,1,0,0,0,0,0,0,0,0)
v_{257}: (0,1,0,0,0,0,0,0,0,1)
v_{258}: (0,1,0,0,0,0,0,0,1,0)
.
.
.
.
v_{767}: (1,0,1,1,1,1,1,1,1,1)

Fig. 1. Sample measurement vectors for class C_1 and class C_2.

The two classes above have identical first- and second-order statistics. Hence, discrimination techniques which rely on such statistics will be useless. Furthermore, there are 1225 second-order statistics for this problem. Examination of this many parameters and incorporation of these in a classifier may be expensive in terms of time and storage.

On the other hand, the PEG procedure will be able to extract from samples of the classes, the significant third-order events. (The *order* of an event is defined as the *number of specified variables.*) Specifically, as the number of samples grows in size, the PEG procedure will generate the following events with a probability approaching one.

Events for Class C_1	
Measurement Number	1,2,···,9,10,11,···,19,20,21,···,29,30,31,···,50
e_1:	(– – ··· – 0 – ··· – 0 – ··· – 0 – ··· –)
e_2:	(– – ··· – 0 – ··· – 1 – ··· – 0 – ··· –)
e_3:	(– – ··· – 1 – ··· – 0 – ··· – 1 – ··· –)
e_4:	(– – ··· – 1 – ··· – 1 – ··· – 0 – ··· –)

Events for Class C_2	
Measurement Number	1,2,3,···,9,10,11,···,19,20,21,···,29,30,31,···,50
e_5	(– – ··· – 0 – ··· – 0 – ··· – 1 – ··· –)
e_6	(– – ··· – 0 – ··· – 1 – ··· – 1 – ··· –)
e_7	(– – ··· – 1 – ··· – 0 – ··· – 0 – ··· –)
e_8	(– – ··· – 1 – ··· – 1 – ··· – 1 – ··· –)

Not only will the events shown above provide accurate classification information for classes C_1 and C_2, the events represent components of a classifier which are economical with respect to the storage which they require.

Another point should be made about statistics which are greater than second order. High-order events may exist which accurately characterize different classes. It may be scientifically helpful not only to discriminate such classes, but also to specify what the different class characteristics are. Such information with regard to disease classes, for example, may point toward better knowledge of causes or cures. Prime events may manifest such characteristics.

The theory of prime events and the PEG procedure for generating estimates of prime events are offered as the basis for a classifier design tool. The effectiveness of this approach depends upon the problem being dealt with and the particular samples which are obtained. Operational characteristics, such as computation time, storage, and

accuracy, need to be stated in terms of the characteristics of a particular problem. Hence, an example of the application of the prime event theory will be described briefly in the next section to show additional practical details.

VII. EXAMPLE APPLICATION

The example application of the theory of prime events discussed here is the design of a classifier for optical character recognition (OCR). The characters are the numerals, 0 through 9, which are printed by a computer-output line printer. The numerals are represented by a 12 × 8 binary array; Fig. 2 shows samples of these numerals. There are relatively large variations in the characters, as may be seen in Fig. 2, and these characters are not members of a font specifically designed for OCR.

The example OCR task was to discriminate the ten classes of numerals discussed above. Furthermore, practical constraints of time and storage were imposed on this design task. Using a specific processor architecture, the classifier was restricted to be a stored-program algorithm which would require no more than 8K bytes of memory. The speed of classification was constrained to classify 500 characters per second.

The above classifier design task, one of a large variety of practical recognition problems which have discrete, type-II variables, was selected as an example because a large group of readers could understand and interpret the results, statistically and visually, without a great deal of interpretation by the author. This design task was a nontrivial multiclass recognition problem which had extremely large dimensional-measurement vectors. The unique features of the prime-event approach, as discussed in the last section, might therefore be manifest.

The sample characters were represented as 96-dimensional binary-valued vectors. An event, therefore, may be viewed as a 96-dimensional ternary-valued vector, and a set of events, the, *definition set* was selected for each class. The method of classification was to assign a sample character to the class whose *definition set* contained the event which covered the sample character.

The practical time and storage constraints described above inhibited straightforward application of prime-event theory. Time is required to compute whether an event covers a sample character and, naturally, the events in the definition sets must be stored in the classifier. To meet the practical constraints in this problem, it was judged that, at most, three events could be stored for each definition set.

Another practical complication of the example problem was that only a finite set of sample numerals was available, thus inhibiting exhaustive statistical analysis. One hundred samples of each numeral were made available for design purposes.

An initial application of the PEG algorithm for the class of 8's, with parameters θ_A and θ_B set to 0 and 1, respectively, yielded one prime-event estimate. Testing this event on samples outside of the design set revealed that over 1 percent of the new 8's were not covered by the

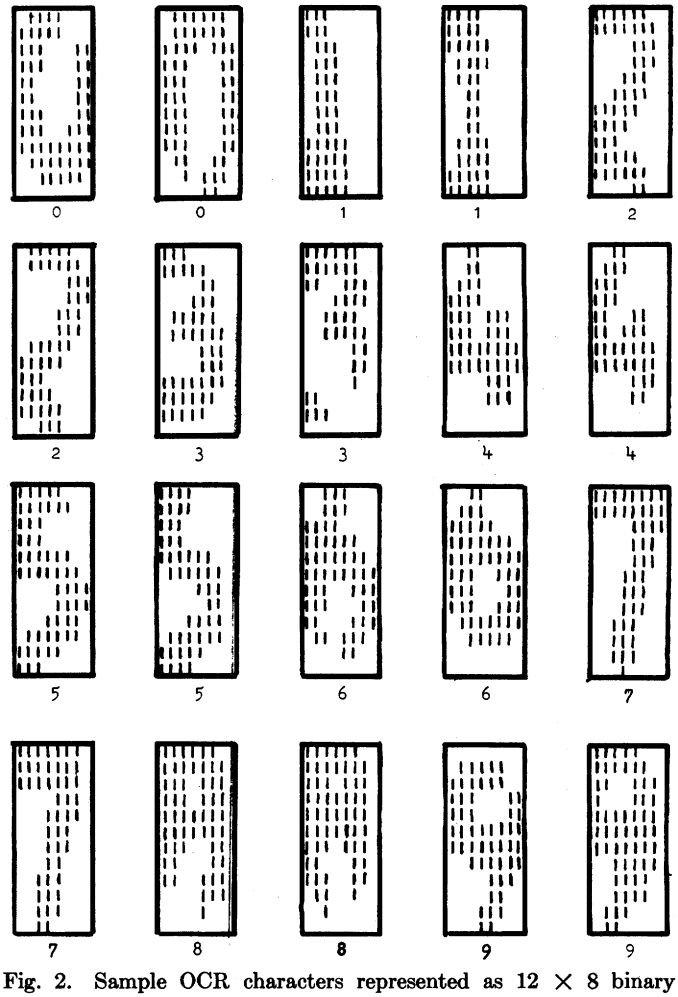


Fig. 2. Sample OCR characters represented as 12 × 8 binary arrays.

generated prime event. The uncovered samples of 8's, however, had a small number of variables (less than five) which differed from the specified variables in the prime-event estimate.

Based on the above results, the concept of covering was expanded to include a measure termed the covering distance. The covering distance from an event to a sample is defined as the number of variable values in the measurement vector which differ from the corresponding specified variables in the event. The rule for classifying a sample was altered to incorporate this principle. The covering distance to each prime-event estimate was computed, and the sample was assigned to the class whose *definition set* contained the event with the smallest covering distance. If the smallest covering distance was greater than five, the sample was rejected.

For the classification procedure stated above, it was determined that 22 events could be stored and incorporated in the software classification algorithm. The goal thus became one of selecting the best 22 events which had covering-distance properties enabling classification accuracy on the order of one substitution error or one rejection of a completed, humanly recognizable character in 10 000 samples. The method for meeting this goal was to increase θ_B when generating prime-event estimates.

(Parameter θ_A was maintained at zero throughout the design.) This resulted in prime events which had covering distances of larger magnitudes to the samples of other classes. This also resulted in larger numbers of prime events being generated.

The large numbers of prime events generated by the PEG algorithm had to be further examined to select a *definition set.* To overcome this design difficulty, a single-list mode of the PEG algorithm, PEG1, which is described in Appendix A, was employed. This procedure generates a subset of the prime-event estimates generated by the PEG algorithm and this subset is guaranteed to cover all samples initially placed in list L. In this way, the definition sets for the ten classes were selected. In Fig. 3, the definition sets which were automatically generated are shown. Table I shows the values of θ_B which were utilized when the definition sets were generated. The results of testing this classifier on, roughly, 9000 samples, which were independent of the design samples, are as follows.

The number of "valid" samples which were correctly classified was 8855. The number of misclassified samples was 0. A number of samples which represented electronic malfunctions, noise, etc. appeared in the test data; these were defined by the designer as reject characters. All eleven of these were correctly rejected by the classifier. No valid characters were rejected.

VIII. ANALYSIS

The procedure of incorporating "masks" or "n-tuples" in an OCR classifier is not a novel technique (see Nagy [34], Bowman [35]). However, the method of determining the specific n-tuples to be incorporated is new. Historically, the n-tuples which were generated were random selections of variables from a select set of variables. Membership in this select set was based upon the fact that maximum-likelihood first-order statistical estimates for a variable value of 0 or 1 exceeded some threshold. The frequency of occurrence of the selected n-tuple within the set of design samples was not known. The estimates of the number n of variables to be specified in the mask was typically based on first-order statistics (Bowman [35]). Furthermore, if two or more n-tuples could be utilized in a classifier, the random selection procedure, alluded to above, provides little assistance in selecting two or more complementary representations of a class.

The method of generating prime-event estimates yields n-tuple masks which have guaranteed characteristics, different from those generated by the random-selection procedure. The prime-event estimates reflect the interdependence of the variables and do not rely solely on first-order statistics. Furthermore, the prime-event estimates represent distinct "modes" of the samples of a class. The frequency of occurrence of the events may be estimated by the percentage of the class samples which the event covers; this value is available in the PEG1 algorithm and was printed out with each generated event. Furthermore, through the use of a selection procedure such as the PEG1 method, multiple masks may be selected which

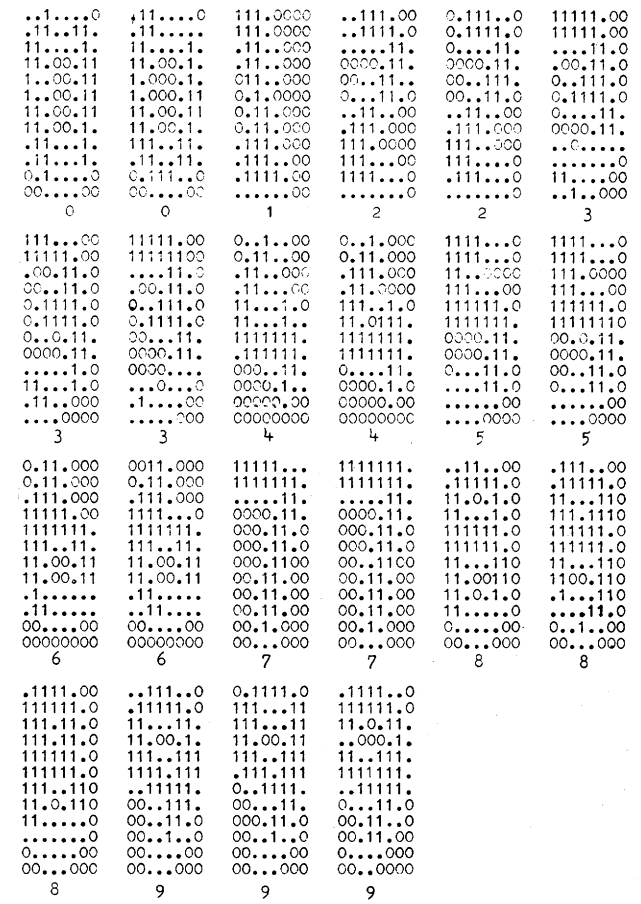


Fig. 3. Automatically generated prime-event estimates arranged as 12 × 8 arrays and depicting the ten definition sets. A "." indicates an unspecified value.

TABLE I
VALUES OF θ_B USED FOR GENERATING EACH DEFINITION SET

Character	θ_B
zero	10
one	11
two	9
three	10
four	10
five	8
six	9
seven	10
eight	9
nine	9

have complementary covering capabilities. These represent superior "fitting properties" of the prime events over the random n-tuple masks. It should also be noted that the prime-event estimates have specific discriminatory properties as defined by parameter θ_B in the generation algorithm. Such properties are neither known nor examined by the random n-tuple selection methods.

Another property of prime events which may be demonstrated visually is their ability to denote the subclasses or the modes of the statistical structure. Evidence of this property was obtained during the classifier design task; but to provide clearer proof of this property, the classes were combined so that only two classes remained. Class C_1

was defined to contain numerals 0 through 4 and class C_2, numerals 5 through 9. Samples of classes C_2 and C_1 were placed in list L and NL, respectively, of the PEG1 algorithm. The algorithm then generated the prime-event estimates, which are shown in Fig. 4, while parameters θ_A and θ_B were set to 0 and 14, respectively. The structure of the numerals 5 through 9 may be seen in the different prime events. Furthermore, some of the prime events in Fig. 4 represent subclasses which are composed of pairs of numerals which have similar graphical representations in areas of the 12×8 array. These pairs of numerals have thus created new subclasses with specific *graphical* structure.

The above example is intended to manifest the method of applying the theory of prime events and the unique properties of the prime-event estimates. The details of the example, the 12×8 array representation of a character, the accuracy of classification, the number of samples utilized, and the potential of this method of OCR, are not at issue here. The OCR classifier design task gives evidence of how the theory of prime events may be applied and of what the results might be.

No special section on caveats for potential users of prime-event theory was included above. It should be understood that all computerized procedures which operate on samples to provide classificatory information require processor time and storage, and that these are functions of the types of samples which are utilized. It is appropriate to note here, however, that the PEG algorithm demanded magnitudes of storage and time which were exponentially related to the number of OCR samples incorporated in the above example. This was in contrast to the PEG1 algorithm, which had an approximately linear demand for time and storage as the number of incorporated samples was varied. The PEG1 algorithm was selected for the OCR classifier design task, and it required less than 60 seconds and 110 000 bytes of storage for any generation of a class *definition set.* (The algorithm was written in Fortran and run on an IBM 370/155.)

IX. CONCLUSION

The problems of designing a classifier for discrete, type-II, variable-measurement vectors have been examined above. The theory of prime events was introduced, and a procedure of generating prime-event estimates was proposed as a tool to aid the classifier designer. An example classifier design task, using a prime-event-generation procedure, was reviewed and the results were examined visually and statistically. This paper introduces a new method of dealing with sample data and provides a new theoretical viewpoint from which to view the classificatory information which has been defined in terms of prime events.

APPENDIX

The following is a description of an additional mode of operation which has been provided for the basic prime event generation algorithm. For some design tasks, a generation procedure which is less exhaustive and faster than the basic prime event generation algorithm may be useful. The PEG algorithm will yield all events which will satisfy the properties of a prime event with respect to the initial lists, L and NL, of samples. The resultant set of estimates of prime events may be "overly complete," "too large," or "too time consuming" to create for some problems. The next design task is to select a practical size set of the resultant events which one estimates will cover all of the chosen class. This also may prove too time consuming for some classification problems.

To cope with these difficulties, it is possible to run the PEG algorithm in a *single list mode.* In this mode, the successive generation of lists is eliminated. Only lists L and NL will be incorporated. Furthermore, this mode of operation will perform the selection of a sufficient set of events to cover the initial list L. This will eliminate the task of comparing covers made up of events generated by PEG.

The following discussion will include a description of the PEG1 algorithm, an example of its operation, and a flow chart of the specific operations in the algorithm. It will be instructive to include the example with the description of the procedure.

The algorithm begins by placing samples of one class in list L and the remaining samples in list NL. As an example, let class C_1 be made up of the following set of four-dimensional, binary-valued vectors. Class C_2, and hence list NL, will be all remaining vectors from the set of four-dimensional binary-valued vectors.

	L	PF	TF	NL
1:	(1 0 1 0)	0	0	(remaining measurement vectors)
2:	(1 0 1 1)	0	0	
3:	(1 1 0 0)	0	0	
4:	(0 1 1 0)	0	0	
5:	(1 0 0 1)	0	0	
6:	(1 0 0 0)	0	0	

PF and TF are vectors of dimension the size of list L and they are initialized to zero.

The basic operations of merging events and checking their ability to cover list NL are retained. The algorithm begins by merging the first pair of samples in list L. Let this be labeled event e_{12}:

$$e_{12} \triangleq (1,0,1,0) + (1,0,1,1) = (1,0,1,—).$$

This event is then checked to see if it covers any member of NL. As shown above, there are two flag vectors, the "temporary flag," TF, and the "permanent flag," PF, which are initialized to zero. These are utilized to keep track of those members of list L which are, or have been, covered by an event previously generated. Event e_{12} covers no member of NL and thus "temporary flags" $TF(1)$ and $TF(2)$ are set to 1. If the event e_{12} did cover some members of NL, then the flags would remain 0.

The next step in the algorithm would be to merge e_{12}

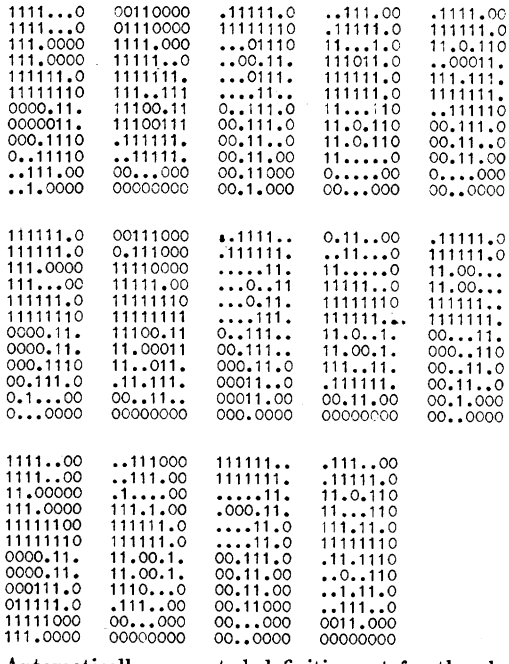


Fig. 4. Automatically generated definition set for the class made up of numerals 5 through 9. A "." indicates an unspecified value.

and measurement vector 3:

$$e_{123} \triangleq (1,0,1,—) + (1,1,0,0) = (1,—,—,—).$$

Checking list NL shows that vector (1,1,0,1) is in NL, and thus e_{123} will not be a retained event since it covers a vector in NL.

Next, form the event e_{124} as above. This too covers some event in NL and is thus not retained. Event e_{125} is then formed: $e_{125} = (1,0,—,—)$. This event covers nothing in NL and hence flag $TF(5)$ is set to 1.

Continuing down list L, the algorithm notes that measurement vector 6 is covered by e_{125}. Thus flag $TF(6)$ is set to 1.

The algorithm now proceeds to the top of list L in an attempt to merge additional vectors. Vector 1 was the "*start vector*" and therefore this "pass" terminates. The permanent flag vector is then updated by forming the pointwide logical "or" of the entries in TF and PF. The event e_{125} is stored, as one of the selected events, in list E, and TF is reset to all 0's. Henceforth, PEG1 will make two "passes" through the samples.

The above procedure is now restarted with one of the measurement vectors not covered thus far. The flag $PF(3)$ is 0 and thus vector 3 becomes the "restart vector." The first step is to merge vector 3 with another vector which is not covered thus far. The flag vectors at this point are as follows:

	1	2	3	4	5	6
PF:	(1	1	0	0	1	1)
TF:	(0	0	0	0	0	0).

From PF one can see that vector 4 is uncovered. Forming, during the first "pass," $e_{34} \triangleq (1,1,0,0) + (0,1,1,0) = (—,1,—,0)$, and checking NL, one notes that (1,1,1,0) is in NL and thus covered by e_{34}. The event e_{34} is disregarded and the algorithm then searches PF for another uncovered vector. If it found one, the result of its merger with vector 3 would be checked to determine if it covered some vector in NL. If no vectors in NL were covered, the search for another 0 in PF would begin again. If the new event covered something in NL, then the event would be disregarded. However, no other 0 was found in PF. The algorithm now attempts to merge vector 3 with any vector in L, not just the uncovered ones. This is referred to as the second "pass."

The event e_{35} is formed and checked against NL. Successfully covering an element in NL, this event is ignored. Event e_{36} is formed and covers no event in NL. However, merging any additional events with e_{36} will yield an event which covers something in NL. Thus, event e_{36} is added to the list E, $PF(3)$ is set to 1, and the next uncovered vector is selected as a restart vector. Two passes are always performed on the list L when generating a potential prime event.

The restart vector is vector 4. Finding no other uncovered measurement vectors, the algorithm proceeds to attempt merging vector 4 with all other elements of list L. Each merger covers something in NL. The result is that vector 4 must be retained in E as the only "potential prime event" covering vector 4. $PF(4)$ is then set to 1.

The algorithm now searches for another uncovered sample in L. All entries in PF are 1, however. When this occurs, the algorithm halts. The set E then contains a subset of the estimated prime events which may be generated by PEG. The resultant set is a sufficient set to cover all the design samples in list L.

The details of the PEG1 algorithm may be obtained from the flow graph of Fig. 5. List L initially contains

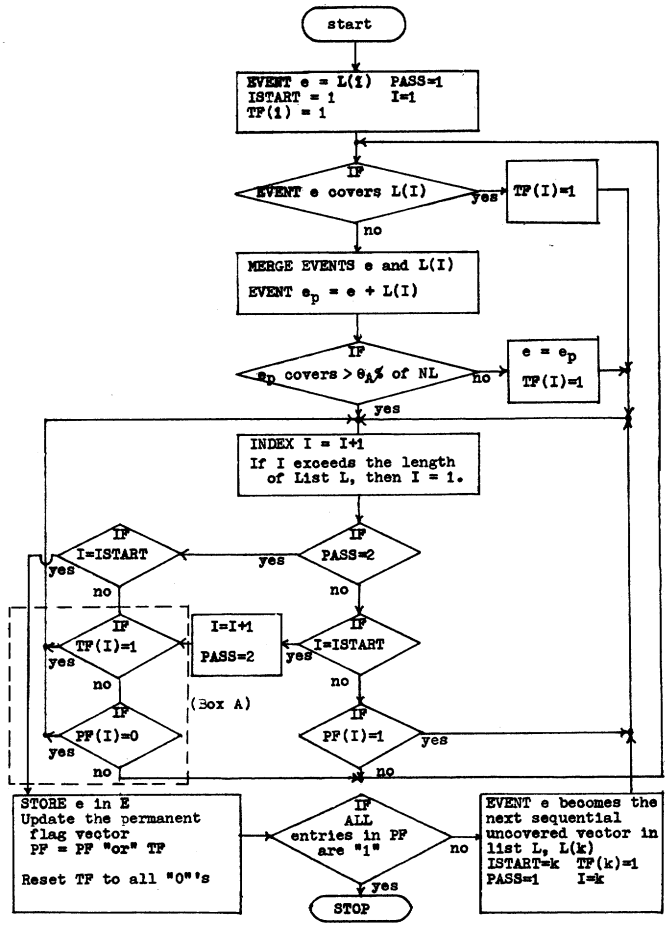


Fig. 5. Flow graph of the PEG1 algorithm.

the samples from the chosen class, and list NL will contain the samples from the other classes. The variable PASS serves to indicate whether the algorithm is attempting a merger with the as yet uncovered samples, PASS = 1, or the covered samples, PASS = 2. Variable ISTART is the index of the member of list L where PEG1 begins its search for mergers. Also, in the PEG1 procedure, as with the PEG algorithm, the definition of "cover" is potentially expandable through the incorporation of parameters θ_A and θ_B, as described previously in Section VI. The purpose of the checking which is performed in Box A is to reduce the amount of redundant merging and checking in the two-pass algorithm.

It should be noted that PEG1 will generate a subset of those events generated by PEG. A sufficient set to cover List L is generated, and this subset may be the same as that generated by PEG.

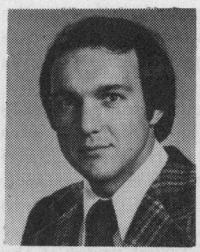
ACKNOWLEDGMENT

The author would like to thank Dr. J. Sammon for supplying the OCR data which were utilized for the example described in Section VII. Furthermore, credit is due Dr. Sammon for stimulating the author's interest in the discrete variable problem discussed in this paper.

REFERENCES

[1] L. N. Kanal, "Interactive pattern analysis and classification systems: A survey and commentary," *Proc. IEEE*, vol. 60, pp. 1200–1215, Oct. 1972.
[2] J. Sammon, "Techniques for discrete, type-II data processing on-line (AMOLPARS)," Rome Air Development Center, Rome, N. Y., Tech. Rep. RADC-TR-71-232, 1971.
[3] L. Kanal, "Adaptive modeling of likelihood classification," Rome Air Development Center, Rome, N. Y., Tech. Rep. TR-66-190, 1966.
[4] D. Foley, "The probability of error on the design set," Ph.D. dissertation, Dep. of Electrical Engineering, Syracuse Univ., Syracuse, N. Y., 1971.
[5] E. Fix and J. Hodges, "Discriminatory analysis," USAF School of Aviation Med., Randolph Field, San Antonio, Tex., Project Rep. 21-49-004, no. 11, 1961.
[6] R. Fisher, "The use of multiple measurements in taxonomic problems," *Ann. Eugen.*, vol. 7, pp. 179–188, Sept. 1936.
[7] J. W. Sammon, Jr., "An optimal discriminant plane," *IEEE Trans. Comput.* (Short Notes), vol. C-19, pp. 826–829, Sept. 1970.
[8] M. Hills, "Discrimination and allocation with discrete data," *Appl. Statist.*, vol. 16, 1967.
[9] G. Lance and W. Williams, "Computer programs for hierarchical polythetic classification (similarity analysis)," *Nature*, vol. 207, p. 159, 1965.
[10] J. W. Sammon, Jr., "Interactive pattern analysis and classification," *IEEE Trans. Comput.*, vol. C-19, pp. 594–616, July 1970.
[11] H. Linhart, "Techniques for discriminant analysis with discrete variables," *Metrika*, vol. 2, pp. 138–140, 1959.
[12] W. G. Cochran and C. Hopkins, "Some classification problems with multivariate qualitative data," *Biometrics*, vol. 17, pp. 11–31, 1961.
[13] P. F. Lazarsfeld, "The algebra of dichotomous systems," in *Studies in Item Analysis and Prediction*, H. Solomon, Ed. Stanford, Calif.: Stanford Univ. Press, 1961.

[14] T. W. Anderson, "On estimation of parameters in latent structure analysis," *Psychometrika*, vol. 12, no. 1, pp. 1–10, 1964.
[15] R. McHugh, "Efficient estimation and local identification in latent class analysis," *Psychometrika*, vol. 21, no. 4, pp. 331–347, 1966.
[16] E. L. Schaefer, "On discrimination using qualitative variables," unpublished Ph.D. dissertation, Univ. of Michigan, Ann Arbor, 1967.
[17] I. J. Good, "Maximum entropy for hypothesis formulation, especially for multidimensional contingency tables," *Ann. Math. Statist.*, vol. 34, pp. 911–934, 1963.
[18] E. S. Gilbert, "On discrimination using qualitative variables," *Amer. Statist. Assoc. J.*, vol. 40, pp. 1399–1413, 1968.
[19] D. R. Cox, *Analysis of Binary Data*. London: Methuen, 1970.
[20] R. R. Bahadur, "A representation of the joint distribution of responses to n dichotomous items," in *Studies in Item Analysis*, H. Solomon, Ed. Stanford, Calif.: Stanford Univ. Press, 1961.
[21] K. Abend, T. J. Hartley, and L. N. Kanal, "Classification of binary random patterns," *IEEE Trans. Inform. Theory*, vol. IT-11, pp. 538–544, Oct. 1965.
[22] C. K. Chow, "A recognition method using neighbor dependence," *IRE Trans. Electron. Comput.*, vol. EC-11, pp. 683–690, Oct. 1962.
[23] C. K. Chow, "A class of nonlinear recognition procedures," *IEEE Trans. Syst. Sci. and Cybern.*, vol. SSC-2, pp. 101–109, Dec. 1966.
[24] R. Sokal and P. Sneath, *Principles of Numerical Taxonomy*. San Francisco: W. H. Freeman, 1963.
[25] D. W. Goodall, "On a new similarity index based on probability," *Biometrics*, vol. 22, pp. 668–670, Dec. 1966.
[26] J. J. Fortier and H. Solomon, "Clustering procedures," in *Item Analysis, Test Design, Classification*, H. Solomon, Ed. Co-op. Res. Project 1327. Stanford, Calif.: Stanford Univ. Press, 1965.
[27] L. McQuitty and J. Clark, "Clusters from iterative intercolumnar correlational analysis," *Educ. Psychol. Meas.*, vol. 28, pp. 2–20, 1968.
[28] J. Elashoff, R. Elashoff, and G. Goldman, "On the choice of variables in classification problems with dichotomous variables," *Biometrics*, vol. 23, pp. 668–670, 1967.
[29] G. T. Toussaint, "Note on optimal selection of independent binary-valued features for pattern recognition," *IEEE Trans. Inform. Theory* (Corresp.), vol. IT-17, p. 618, Sept. 1971.
[30] R. Prather, *Introduction to Switching Theory*. Boston: Allyn and Bacon, 1967.
[31] Y. C. Ho and R. L. Kashyap, "An algorithm for linear inequalities and its application," *IEEE Trans. Electron. Comput.*, vol. EC-14, pp. 683–688, 1965.
[32] T. M. Cover, "Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition," *IEEE Trans. Electron. Comput.*, vol. EC-14, pp. 326–334, June 1965.
[33] K. Fukanaga and W. L. G. Koontz, "Application of the Karhunen–Loève expansion to feature selection and ordering," *IEEE Trans. Comput.*, vol. C-19, pp. 311–318, Apr. 1970.
[34] R. Casey and G. Nagy, "Recognition of printed Chinese characters," *IEEE Trans. Electron. Comput.*, vol. EC-15, pp. 91–101, Feb. 1966.
[35] R. M. Bowman, "On n-tuple optimization and *a priori* error specification for minimum distance pattern classifiers," Ph.D. dissertation, Dep. Elec. Eng., Univ. of Virginia, Charlottesville, 1970.



James C. Stoffel (S'64–M'72) was born in Rochester, N. Y., in 1946. He received the B.S.E.E. degree from the University of Notre Dame, Notre Dame, Ind., in 1968 and the M.S. and Ph.D. degrees from Syracuse University, Syracuse, N. Y. in 1970 and 1972, respectively.

From 1968–1972, he worked in analog and digital signal processing and pattern recognition as a consultant to Microwave Systems, Inc. Since 1972 he has been with the Research Laboratories Department, Xerox Corporation, Rochester, N. Y His research interests include picture and waveform processing, pattern recognition, and digital communications. He is an associate editor of the Journal of the Pattern Recognition Society.

Dr. Stoffel is a member of Tau Beta Pi, Eta Kappa Nu, and the Association for Computing Machinery.

Correspondence

Parallel Balancing of Binary Search Trees

SHI-KUO CHANG

***Abstract*—A method for the balancing of binary search trees on a highly parallel computer is described. This method can be adapted for execution on a multiprocessor computer system.**

***Index Terms*—Binary search trees, highly parallel computer, information retrieval, parallel computation, sorting.**

Manuscript received February 28, 1973; revised November 8, 1973.
The author is with the IBM T. J. Watson Research Center, Yorktown Heights, N. Y. 10598.

I. INTRODUCTION

This correspondence describes a method for the parallel balancing of binary search trees. The method is first developed with respect to a highly parallel computer having a very large number of processors and parallel-addressable memory locations. It is then adapted for execution on a multiprocessor computer system. Simulation results using a multiprocessor model are discussed.

Binary search trees have long been recognized as useful structures for storing and retrieving data, particularly when sequential processing of data is also required. In many applications, records (data items) to be stored in a search tree are not available simultaneously, but are received one by one in some arbitrary order. The tree is constructed dynamically as the records come in. We shall assume that each record has a numerical key and the tree is to be constructed