N-Tuple Features for OCR Revisited

Dz-Mou Jung, *Member, IEEE Computer Society*, M.S. Krishnamoorthy, George Nagy, and Andrew Shapira

Abstract—N-tuple features for optical character recognition have received only scattered attention since the 1960s. Our main purpose here is to show that advances in computer technology and computer science compel renewed interest. N-tuple features are useful for printed character classification because they indicate the presence or absence of a given rigid configuration of n black and white pixels in a pattern. Desirable n-tuples fit each pattern of a specified (positive) training set of characters in at least p different shift positions, and fail to fit each pattern of a specified (negative) training set by at least $n - q$ pixels in each shift position. In this work we prove that the problem of finding a distinguishing n-tuple is NP-complete, by examining a natural subproblem with binary strings called the missing configuration problem. The NP-completeness result notwithstanding, distinguishing n-tuples are found automatically in a few seconds on contemporary workstations. We exhibit a practical search algorithm for generating, from a small training set, a collection of n-tuples with low class-conditional correlation and with specified design parameters n, p, and q. The generator, which is available on the Internet, is empirically shown to be effective through a comparison with a benchmark generator. We show experimentally that the design parameters provide a useful tradeoff between distinguishing power and generation time, and also between the conditional probabilities for the positive and negative classes. We explore the feature probabilities obtainable for various dichotomies, and show that the design parameters control the feature probabilities.

Index Terms—Backtracking, character features, classification, decision trees, distinguishing string, missing configuration, n-tuples, OCR, simulated parallelism.

1 Introduction

> We have an interest in decision-making circuits with the following qualities: 1) measurable high reliability in decision making, 2) either a high or a low reliability input, and 3) possibly low reliability components [2].

With these words in 1959, Bledsoe and Browning introduced n-tuples ("decision-making circuits") for recognizing typewritten characters, hand-blocked print, and handwritten script.

For our purposes, an n-tuple is simply a collection of n pixels with distinct locations. Fig. 1 shows a 7-tuple T_1 comprising four black pixels and three white pixels. The tuple "fits" the c exemplar and the e exemplar, and does not fit the n exemplar or the r exemplar. That is, if we copy T_1 onto a transparency and superimpose the transparency onto the c, it is possible to shift the transparency so that each pixel of the 7-tuple is the same color as the corresponding pixel of the c. Here each character is considered to be embedded in a sea of white pixels, and the tuple is moved only by shifting, and not, for example, by rotating or reflecting. In this way, we find that T_1 fits the c and e, but does not fit the n or r. We say that the 7-tuple T_1 is designed for the $ce - nr$ dichotomy.

In the early sixties, researchers at IBM's T.J. Watson Research Center conducted a massive series of experiments, combining n-tuple features with various types of statistical classifiers for high-performance typewritten and hand-printed character recognition [1], [6], [7], [9]. Simple hardware can determine whether or not a given n-tuple fits a given character, so n-tuples were sometimes called *decision-making circuits*. N-tuples were mentioned only briefly in Levine's 1969 survey of feature extraction [8], but Nadler praised them in his 1972 State of the Art in Optical Character Recognition [10]. The sensitivity of n-tuples to sample size was investigated at the UK National Physical Laboratories [20], and Stentiford exploited n-tuple independence for reading printed postal addresses in 1985 [19]. Since then, the quest for magic universal features has meandered in other directions. Our main purpose here is to show that advances in computer science and computer technology compel serious reexamination of the applicability of n-tuples to OCR.

In this reprise of an earlier strand of research, we present a generator that finds a collection of distinct n-tuples for any specified dichotomy of binary character patterns. The generator is intended for the automatic construction of OCR systems where little or no prior information is available about the nature of the symbol shapes. We prove that the problem of finding a distinguishing tuple is NP-complete, by showing that a natural subproblem with binary strings called the *missing configuration problem* is NP-complete. The NP-completeness of tuple generation suggests that in general, generating tuples may be difficult. Despite this, for practical problems the generator finds distinguishing n-tuples in a few seconds on contemporary workstations. The quality of the generator is established by looking at absolute execution times, and by comparing the generator to a simple benchmark generator. We give a set

- *D.-M. Jung is with Caere Corporation, 100 Cooper Court, Los Gatos, CA 95030. E-mail: jung@caere.com.*
- *M.S. Krishnamoorthy, G. Nagy, and A. Shapira are with Rensselaer Polytechnic Institute, Troy, NY 12180. E-mail: {moorthy, nagy, shapira}@ecse.rpi.edu.*

Manuscript received June 26, 1995; revised Apr. 15, 1996. Recommended for acceptance by J.J. Hull.
For information on obtaining reprints of this article, please send e-mail to: transpami@computer.org, and reference IEEECS Log Number P96043.

of design parameters that provide control of the class-conditional feature probabilities associated with a given dichotomy and n-tuple. We also present statistical evidence that for a given dichotomy, tuples can be generated that are not highly correlated and can therefore be used with simple classification methods.

The n-tuple generators in this paper are part of a C language software library for generating n-tuples. This library is available on the Internet [16].

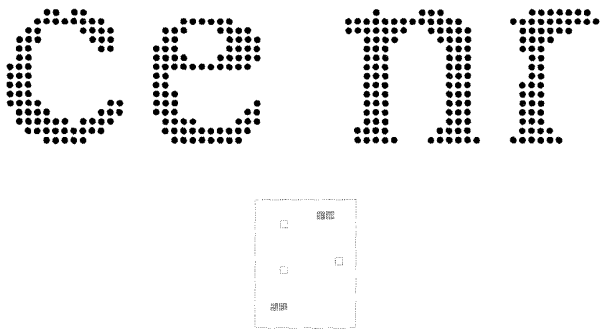


Fig. 1. A 7-tuple T_1 that fits the *c* and *e* and does not fit the *n* or *r*.

Returning to Fig. 1, if the exemplars (the training samples) are representative of other instances of *c*, *e*, *n*, and *r*, then we can expect the tuple T_1 to behave similarly on new exemplars (the test samples). We say that T_1 is designed to fit the positive class C^+ of *c* and *e*, and to misfit the negative class C^- of *n* and *r*. Such a tuple is an elementary two-category classifier. Several tuples can be used in combination to build multicategory classifiers, or to improve the accuracy of two-category classifiers. Minimum-distance, weighted linear network, quadratic, nearest neighbors, and decision tree classifiers have all been used successfully in conjunction with n-tuples.

A tuple is associated with the presence or absence of a specific configuration of black and white pixels in a given pattern. At one extreme, a 2-tuple detects a pair of pixels of given color and relative displacement. At the other extreme, a tuple with as many elements as the entire pattern array constitutes a matched filter (or mask, or template) for a specific binary pattern. For OCR, tuples are sought that match most instances of several designated alphanumeric pattern classes, and do not match most instances of other designated classes.

Given a set of character classes, it is easier to find n-tuple features for some dichotomies than for others. For instance, the *ceo-rnm* dichotomy is easier than the *rnc-eom* dichotomy. The formulation of suitable dichotomies, which is a difficult problem in itself, is addressed in [21].

We are interested in n-tuples because they are uniquely suited for the automated design of single-font OCR systems when only a small number of labeled samples are available. Limited sample availability is characteristic of in-the-field adaptation. We consider field adaptation desirable, even to only a single document. For a given document, an adaptive system using features that discriminate between the classes of the document's actual typeface can perform at least as well as a static multifont system. Possibly, the adaptive system can do better.

Accordingly, throughout this paper we make the following assumptions:

- only one font is to be recognized,
- only a small number of labeled samples are available, and
- the dichotomy is given.

1.1 Classification Using N-Tuples

Regardless of the particular type of classifier used, the error rate is closely linked to the class-conditional feature vector probabilities [3]. These probabilities are

$$P(\mathbf{x} \mid C^+) = P(x_1, x_2, \ldots, x_r \mid C^+) \text{ and } P(\mathbf{x} \mid C^-) = P(x_1, x_2, \ldots, x_r \mid C^-).$$

Here r is the length of the feature vector $\mathbf{x}$, and each element x_i is an indicator random variable that is true iff feature i is present in the character under consideration. If x_i is true we can write the logical statement x_i = true, or simply x_i; if x_i is false we write $\overline{x_i}$. In our case, i identifies a tuple we are interested in, and x_i indicates whether the tuple fits the character under consideration.

For accurate classification, we want to maximize the marginal probabilities $P(x_i \mid C^+)$, and minimize the marginal probabilities $P(x_i \mid C^-)$. That is, we would like

$$P(x_i \mid C^+) \approx 1 \text{ and } P(x_i \mid C^-) \approx 0, \text{ for all } x_i.$$

For accurate classification and ease of classifier design, we want the individual features x_i to be class-conditionally independent; we would like the following to hold for all feature vectors $\mathbf{x}$:

$$P(\mathbf{x} \mid C^+) = P(x_1 \mid C^+)P(x_2 \mid C^+) \cdots P(x_r \mid C^+), \text{ and} \quad (1)$$

$$P(\mathbf{x} \mid C^-) = P(x_1 \mid C^-)P(x_2 \mid C^-) \cdots P(x_r \mid C^-). \quad (2)$$

Given a small training sample of representative patterns, we wish to generate tuples that have the above properties. Our scheme is based on the following propositions.

P1 Small values of n are most desirable for generating tuples that resist variations in positive test patterns.

P2 Large values of n are most desirable for generating tuples that resist variations in negative test patterns.

P3 The probability of a generated tuple fitting a positive test pattern increases with p, if we generate tuples that fit each positive training exemplar in p or more shift positions.

P4 The probability of a generated tuple fitting a negative test pattern decreases as q decreases, if we generate tuples that agree with each negative training exemplar at q or fewer pixels in each shift position.

P5 Tuples with appropriate values of (n, p, q) can be found without using excessive computing resources.

The validity of these propositions is demonstrated in subsequent sections.

The first two propositions imply a trade-off between the two conditional probabilities that depends on n. Our experiments support earlier findings [6] that $n = 7 \pm 2$ is usually acceptable. In view of the third and fourth propositions,

for a specified value of n, for each q we select the largest p for which tuples can be found. Different (p, q) combinations bias reliability towards either the positive or negative class. For each dichotomy, we can generate tuples according to (n, p, q) values appropriate for the dichotomy's class probabilities.

1.2 Outline

An outline of the remainder of the paper is as follows. In Section 2, we discuss the p and q parameters and the order that these parameters impose on the set of all tuples. Section 3 shows that the tuple generation problem is NP-complete. Section 4 describes the tuple generator and explores the generator's speed and the relation between speed and the design parameters. In Section 5, we investigate the optimal tuple size; the dependence of the feature probabilities on the design parameters; the statistical correlation of the generated tuples; and the effect of the choice of dichotomy on the feature probabilities. We conclude with a discussion of the implications of our improved method of tuple generation for printed character classification.

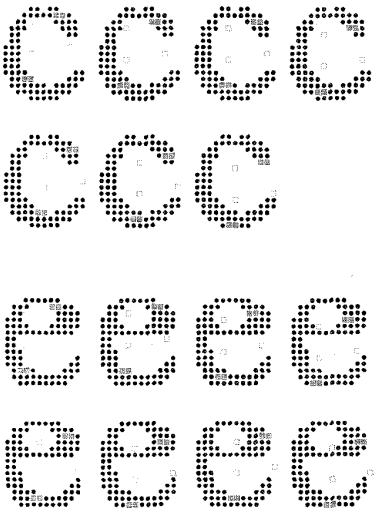


Fig. 2. The matching positions of a 7-tuple T_1 for two characters.

2 p-q Stability

Before discussing the p and q parameters, let us solidify a few concepts.

Earlier, we stated that an n-tuple is a set of n pixels at distinct locations. Here we define a *pixel* to be an integer triple (r, c, v), where the row r and column c give the pixel's location, and $v \in \{0, 1\}$ gives the pixel's color (0 is white and 1 is black). The *order* of a tuple T is the number of pixels $|T|$.

A *character* E with r_{max} rows and c_{max} columns is an array $A[1 \ldots r_{max}, 1 \ldots c_{max}]$ with elements having value 0 or 1. We embed E in a sea of white pixels by defining an auxiliary array $B[-\infty \ldots +\infty, -\infty \ldots +\infty]$, so that $B[i, j] = A[i, j]$ if $1 \le i \le r_{max}$ and $1 \le j \le c_{max}$, and $B[i, j] = 0$ otherwise. A tuple T *fits* E (exactly) if there exists some *shift* $(\Delta r, \Delta c)$ such that for each pixel $(r, c, v) \in T$, $B[r + \Delta r, c + \Delta c] = v$.

For the remainder of this section let us fix a set E^+ of positive class exemplars, a set E^- of negative class exemplars, and a tuple size n.

Given a tuple T, we define p_T to be the maximum value p such that for each character $E \in E^+$, there are at least p shifts that give an exact fit between T and E. Similarly, we define q_T as the minimum value q such that for each character $E \in E^-$, for no shift do T and E agree in more than q pixels. A tuple T is said to be *p-q-stable*, or to *satisfy the* (p, q) *constraint*, if $p_T \ge p$ and $q_T \le q$. We direct our attention to tuples T with both $p_T \ge 1$ and $q_T \le n - 1$.

The idea behind p-q-stability is that the larger p_T is, the more T resists noise in positive test samples; the smaller q_T is, the more T resists noise in negative test samples. Fig. 2 shows the characters c and e and the 7-tuple T_1 from Fig. 1. For each of the two characters, T_1 is shown superimposed in all positions for which a match exists. This shows that $p_{T_1} = 7$. Illustrating q_{T_1} is more difficult. In this example, $q_{T_1} = 5$, i.e., when T_1 is superimposed in each shift position on either of the negative characters n or r, there are always at least $7 - 5 = 2$ pixels of T_1 that mismatch the character.

The matches in Fig. 2 all involve one or more pixels at the edge of a character. This is to be expected, since the characters have a small stroke width. For sufficiently wide characters, however, it is desirable for tuples not to depend on edge pixels having a particular value. Edge pixels are inherently unstable due to random-phase sampling noise—the noise from the unpredictable position of the scanner sampling grid with respect to a printed character. The Gen1 generator described in this paper has been used to reduce the error rate by generating p-q-stable tuples that do not depend on any edge pixel having a particular value [11], [21].

We say a tuple T *dominates* a tuple U if $|T| = |U|$, $p_T \ge p_U$, $q_T \le q_U$, and at least one of the inequalities holds strictly. If T dominates U, then in the absence of other knowledge, T is to be preferred to U as a classification feature, assuming Propositions P3 and P4 are true. The domination relation is transitive and imposes order on the tuples of order n.

Still using E^+, E^-, and n, let us define a Boolean function $f(p, q)$ that is true iff there exists a tuple satisfying (p, q). If a tuple satisfies (p, q), then it also satisfies $(p - 1, q)$ and $(p, q + 1)$. The following, then, are fundamental properties of the p–q space:

$$f(p, q) \rightarrow f(p - 1, q) \text{ and } f(p, q) \rightarrow f(p, q + 1). \quad (3)$$

The staircase shape of f is illustrated by the following hypothetical example, with dots standing for "false" and 1s for "true."

		q					
	f	1	2	3	4	5	6
	1	.	1	1	1	1	1
	2	.	1	1	1	1	1
	3	.	.	.	1	1	1
p	4	.	.	.	.	1	1
	5	.	.	.	.	.	1
	6	.	.	.	.	.	1
	7	.	.	.	.	.	1
	8	.	.	.	.	.	.

We call a (p, q) value *critical* if $f(p, q)$ is true and $f(p + 1, q)$ and $f(p, q - 1)$ are both false. A tuple is *critical* if it satisfies a critical (p, q) value. The set of critical values completely specifies f. For the hypothetical example above, the critical values are (2, 2), (3, 4),(4, 5), and (7, 6).

Because of the dominance relation, critical tuples are the ones we want to find. Prior to a search, we do not know what the critical (p, q) values are. Indeed, the next section tells us that in general, determining the critical values may be computationally intractable.

3 Complexity Analysis

In this section we consider the computational complexity of finding tuples. Here, we show that the problem of finding tuples is NP-complete. From this result, most complexity theorists would conclude that no algorithm finds tuples quickly for all dichotomies. Still, an algorithm may exist that finds tuples quickly for all dichotomies that arise in OCR systems.

We show that tuple-finding is NP-complete by proving that a one-dimensional special case called the *missing configuration problem* (or the MC problem) is NP-complete. The MC problem is a simply-stated problem with binary strings and may be applicable to areas other than OCR; for example, to DNA string problems. Let us now examine this problem.

A *configuration* C is a set of integers $C = \{c_1, c_2, \ldots, c_n\}$, where $1 = c_1 < c_2 \ldots < c_n$. We say C has *order* n and *span* c_n. For example, {1, 3, 6} is an order 3 configuration with span 6. Another way to write this configuration is $1x1xx1$.

Informally, the missing configuration problem is to find, given a binary string S and positive integers n and r, an order n configuration C with span at most r, such that C is missing from S. For example, the order 3 configuration $\{1, 3, 6\} = 1x1xx1$ is missing from the string 1010100011111, but is present in the string 111010110111. We treat the xs in $1x1xx1$ as don't-care elements when determining whether $1x1xx1$ matches a given string. We now precisely state the MC problem and show that it is NP-complete.

PROBLEM INSTANCE. A string $S = s_1 s_2 \ldots s_{|S|} \in \{0, 1\}^*$, positive integers n and r, $n \le r \le |S|$.

QUESTION. Does there exist an order n configuration with span at most r that appears nowhere in S, i.e., is there a set of integers $C = \{c_1, c_2, \ldots, c_n\}$, $1 = c_1 < c_2 < c_3 < \cdots < c_n \le r$, such that for all i, $0 \le i \le |S| - c_n$, there exists some $c \in C$ with $s_{c+i} = 0$?

THEOREM 1. *The missing configuration problem is NP-complete.*

PROOF. It suffices to show that

1) MC is in NP, and
2) some NP-complete problem reduces to MC in polynomial time.

The first part follows easily. Given a configuration that solves a given MC instance, we can test the configuration in each shift position to verify in polynomial time that the configuration appears nowhere in the given string. We prove the second part by reducing the known NP-complete "set cover" problem to MC. Our formulation of the set cover problem is derived from Garey and Johnson [4]:

PROBLEM INSTANCE. Universe $U = \{1, 2, \ldots, |U|\}$, a set $D = \{D_1, D_2, \ldots, D_{|D|}\}$, where each D_i is a subset of U, and a positive integer $k \le |D|$.

QUESTION. Does D contain a cover for U of size k, i.e., is there a subset $D' \subseteq D$ with $|D'| = k$ such that every element of U belongs to at least one member of D'?

Now suppose U, D, and k form an instance of the set cover problem. From this we construct an instance of the MC problem in polynomial time, as follows. Set $r = |D| + 2$ and $n = k + 2$. For each $u \in U$, we construct a string $Y_u = y_{u,0} y_{u,1} \ldots y_{u,r-1}$, where $y_{u,0} = y_{u,r-1} = 1$, and for all i, $1 \le i \le |D|$, $y_{u,i} = 0$ if $u \in D_i$ and $y_{u,i} = 1$ otherwise. Let 0^r denote the string of r 0s, and 1^{r-1} the string of $(r - 1)$ 1s. We form the string S by concatenation:

$$S = Y_1 0^r Y_2 0^r \cdots Y_{|U|} 0^r 1^{r-1}.$$

An example reduction is shown below.

Instance of Set Cover Problem
$U = \{1, 2, 3, 4, 5\}$, $k = 2$,
$D = \{D_1, D_2, D_3, D_4\}$, where $D_1 = \{1, 2, 3\}$,
$D_2 = \{3, 4\}$, $D_3 = \{4, 5\}$, $D_4 = \{1, 2\}$.

Instance of Missing Configuration Problem
$r = 6$, $n = 4$,
$S = 101101\ 000000\ 101101\ 000000$
$100111\ 000000\ 110011\ 000000$
$111011\ 000000\ 11111$.

Here, the set $\{D_1, D_3\}$ is a cover for U, and the corresponding solution to the given instance of the MC problem is the configuration {1, 2, 4, 6}.

Returning to the general case, we must show that there is a solution to the MC instance iff there is a solution to the set cover instance. We consider the forward direction first.

Let C be a solution (with $n = k + 2$ ones) to the MC instance. Since the problem requires that C have span r or less, and S contains a substring of $(r - 1)$ consecutive 1s, we know that the span of C is exactly r. So we can write $C = \{1, i_1, i_2, \ldots, i_k, r\}$, where $1 < i_1 < i_2 < \ldots < i_k < r$. Set $D' = \{D_{i_1}, D_{i_2}, \ldots, D_{i_k}\}$. We now show that D' is a solution to the set cover instance by exhibiting, for each $u \in U$, a member of D' that contains u. Fix u. Since C does not fit S, in particular C does not match S when C and Y_u are aligned. Because $y_{u,0} = y_{u,r} = 1$, there must exist some m, $1 \le m \le |D|$, such that $y_{u,m} = 0$ and $m + 1 \in C$. This implies that $u \in D_m$ and $D_m \in D'$.

For the other direction, let $D' = \{D_{j_1}, D_{j_2}, \ldots, D_{j_k}\}$ be a solution to the set cover instance. We now show that the following order n configuration is a solution to the MC instance: $C = \{1, j_1 + 1, j_2 + 1, \ldots, j_k + 1\ r\}$. The only shifts for which C matches S are when C and Y_u are

aligned, for some Y_u. We will be done if we can show, for each Y_u, that C and Y_u do not match. Fix Y_u. Since D' covers D, there exists some $D_m \in D'$ such that $u \in D_m$. This means that $y_{u,m} = 0$ and $m + 1 \in C$. Thus C and Y_u do not match. □

The MC problem can be solved by brute force in $O\left(\binom{r-1}{n-1}(|S| - n + 1)\right)$ time. The quantity $\binom{r-1}{n-1}$ is maximized when $n - 1 = \lfloor (r-1)/2 \rfloor$, so for fixed r the time is $O(|S|)$.

The problem without the span limit r is solvable for $n \geq 3$ in $O(|S|)$ time, as follows. Let $S = s_1 s_2 \ldots s_{|S|}$. When S consists of all 1s, S has no missing configuration. Otherwise, if $s_1 = 0$ or $s_{|S|} = 0$ then any configuration of span $|S|$ is missing from S. If $s_1 = s_{|S|} = 1$, then take as a missing configuration any configuration of span $|S|$ that has $\{1, z, |S|\}$ as a subset, where $z \in \{2, 3, \ldots, |S| - 1\}$ is chosen arbitrarily such that $s_z = 0$.

There are many generalizations of the MC problem, including allowing multiple strings, arrays of higher dimension instead of strings, larger alphabets, and variations like $p \neq 1$ or $q \neq n - 1$ in the tuple finding problem for OCR. One version is a 1-d subproblem of the tuple finding problem. In this problem we are given n, a positive exemplar E, and a negative exemplar F, and we want to find a tuple T comprising n black and/or white pixels, such that T that has span at most $r = |E|$, fits E, and does not fit F. If E consists solely of black pixels, then we have exactly the MC problem. Thus the MC problem is a subproblem of the tuple finding problem, and the tuple finding problem is NP-complete.

Here, the bits of the negative exemplar are part of the reduction, whereas the only part of the positive exemplar that is used is its length. Also, for a given value of n, it is trivial to determine whether or not a tuple exists that fits the positive exemplar. If n is larger than the size of the positive exemplar, then no such tuple exists; otherwise, a solution can be obtained by taking any n pixels of the positive exemplar. These two facts suggest that in our formulation the difficult aspect of tuple generation lies in satisfying the q constraint.

4 Generating N-Tuples

Originally, we tried generating solution tuples at random. But as problem instances become increasingly constrained through higher values of p, lower values of q, more difficult dichotomies, and more characters per class, random tuple generation becomes infeasible. Random tuple generation found no tuples at all for several dichotomies with $q = n - 3$. The failure of random tuple generation led us to consider more sophisticated search methods.

We generate tuples using a backtracking algorithm called *Gen1*. This generator has been used reliably for hundreds of CPU-hours in OCR studies [5], [11], [15], [21].

We also use a generator called *Gen0*. Gen0 is a relatively naive backtracking algorithm whose main purpose is to serve as a benchmark to help measure the performance of Gen1.

Section 4.1 discusses common aspects of Gen0 and Gen1. Sections 4.2 and 4.3 cover aspects specific to Gen0 and Gen1, respectively. (The discussion of Gen0 includes concepts used for Gen1.) Section 4.4 summarizes experiments with the generators.

Our focus here is on aspects of Gen0 and Gen1 that are fundamental to Gen1's operation or are relevant to OCR. Some other aspects of the generators are discussed in [17].

An open problem concerning tuple generation is to give a necessary and sufficient condition for a dichotomy to admit a distinguishing tuple of given order. Given the NP-completeness result, the formulation of such a condition may be impossible. We are still investigating the problem for the special case of printed characters.

In our OCR experiments, we have not encountered any dichotomy that is known not to have some distinguishing tuple. But certainly such cases are easy to find, if we allow (p, q) to be specified. As the (p, q) constraint is made more restrictive, eventually we reach a point where no distinguishing tuple can be generated. The most desirable trade-off between search time and quality of (p, q) is application dependent and cannot be settled in a general way. One option is to gradually relax the (p, q) constraint until a tuple is found in the allotted time. In the design of our decision tree classifiers, a different strategy was used: if a tuple satisfying the required (p, q) could not be found, another generation attempt was made, using a different subset of the training sample.

4.1 Common Aspects of Gen0 and Gen1

Gen0 and Gen1 input, or a *problem instance*, contains positive and negative character class exemplar sets E^+ and E^-, and a value of n. For a given problem instance, we restrict our attention to tuples whose pixels are drawn from a pixel set Π, where

$$\Pi = \left\{(r, c, v) : 1 \leq r \leq \min_{E \in E^+} \text{rows}_E,\ 1 \leq c \leq \min_{E \in E^+} \text{cols}_E, v = 0 \text{ or } 1\right\}.$$

Here the dimensions of a given character E are denoted by rows_E and cols_E. If desired, the caller can instruct Gen0 or Gen1 to draw tuple pixels from a pixel set other than the smallest bounding box Π of the positive characters.

The selection of the underlying pixel set can affect the quality, search time, and existence of tuples. Two reasons for choosing Π as the allowed pixel set are as follows. First, if a tuple is to be applied to a character embedded in a document, and the tuple does not fit in the character's bounding box, then the tuple's performance may be degraded by the presence of nearby characters in the document. Also, using Π instead of a larger pixel set keeps the search space (and search time) relatively small.

The generators can run in one of two modes. In the first mode, the generator looks for a tuple satisfying a (p, q) pair specified by the caller of the generator. In this mode, the generator returns a tuple satisfying the specified (p, q) pair, or "not found" if a resource allotment is exceeded before a solution is found.

The second mode operates by repeatedly invoking the

first mode to find a set of approximately critical tuples, as shown below.

```
procedure pqselect (integer n)
  set of tuples L initialized to empty
  integer p initialized to 0, q initialized to 1
  while (q ≤ n − 1) do
      search(p + 1, q)    // search seeks a tuple
                          // satisfying p + 1, q.
      if (The search found a tuple T) then
        while (T satisfies (p + 1, q)) do
            p ← p + 1
        L ← L ∪ {T}
      else
        q ← q + 1
  while (Some tuple T ∈ L is dominated by some tuple
         U ∈ L) do
      Remove T from L.
  return L
```

In the first loop, we invoke the first mode for various (p, q) pairs, keeping to the perimeter of a staircase-shaped region as described in Section 2. The tuple dominance relation justifies using the (more efficient) perimeter search instead of searching over the area of the staircase.

Gen0 and Gen1 both use backtracking to look for tuples satisfying a given (p, q) pair. Backtracking can be used because of the following fact: if T and U are tuples such that $U \supset T$ and T fails the p or q constraint, then U also fails the p or q constraint. A failed tuple cannot be made successful by adding pixels to it.

Two operations occur frequently during the search:

1) In Gen0 and Gen1, determine whether a given tuple T meets the p and q constraints, and
2) In Gen1, evaluate a tuple according to a certain evaluation function (see Section 4.3).

According to execution profiling, during one execution the above operations accounted for 95% of Gen1's time. Some effort was spent making these operations fast in Gen0 and Gen1 by using bit arrays and bit-parallel operations like word-AND.

Consistent with their backtracking nature, Gen0 and Gen1 have modest memory requirements.

4.2 Gen0

Gen0 is a relatively naive backtracking search of the *tuple tree for* Π. For a given n, the tuple tree is defined relative to a given total order $\lambda(y)$ on the pixels $y \in \Pi$. The tree nodes at level i, $0 \le i \le n$, are i-tuples. The children of a tuple T are the $(|T| + 1)$-tuples of the form $T \cup y$, where $y \in \Pi$ is a pixel with $\lambda(y) > \max_{t \in T} \lambda(t)$. For a given order λ and value of n, the tuple tree is well-defined and includes each i-tuple exactly once, for $0 \le i \le n$.

The *search tree* for Gen0 is a subtree of the tuple tree. The subtree depends on the given problem instance, p, q, and λ. Gen0 selects λ randomly with each order equally likely. The search tree is implicitly formed from the resulting tuple tree by pruning nodes that fail the p or q constraint. Gen0 carries out a backtracking (depth-first) search of the search tree. For each pixel location, the two pixels sharing the given location are searched in a random fixed order.

An example search tree is shown in Fig. 3. For simplicity, the figure omits tuples that have white pixels. (Incidentally, in this example, no solution has white pixels.)

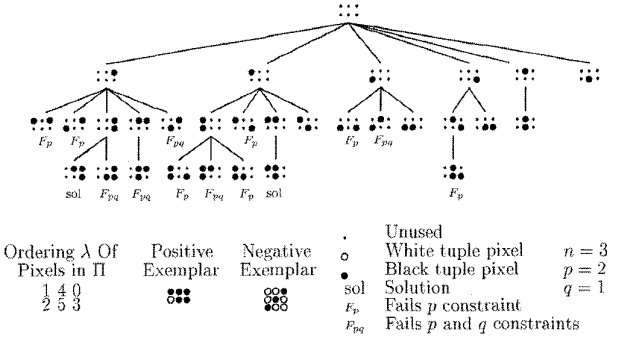


Fig. 3. Example search tree for Gen0 (black-only tuples).

The random selection of λ enables Gen0 to find different solutions on successive executions. Also, this results in a faster search than, say, making λ correspond to a row-major order of Π, at least in part because a random order reduces the likelihood of searching consecutively through many tuples clustered in a small area. The speed-up is of secondary importance, since Gen0 is only a benchmark.

Gen0 may process tuples that are translates of each other. This is perhaps undesirable, but only increases execution time significantly if many branches of the search tree are exhausted, i.e., the whole tree can be searched reasonably quickly. For practical problems this does not occur, except conceivably when n is small. We looked at some dichotomies with $n = 7$ and in no case did Gen0 process isomorphic tuples.

4.3 Gen1

Gen1 uses two search parameters t and w that are supplied by the caller. Gen1 differs from Gen0 primarily in three interrelated ways.

1) Tuple equivalence classes are used to reduce the size of the search space. If two or more tuples are translates of each other, then Gen1 visits at most one of them.
2) Gen1 uses simulated parallelism, a general technique that can reduce the search time on a single processor when solutions are distributed nonuniformly across search regions [12]. Running on one processor, Gen1 uses timeslicing to simulate a parallel search across selected search regions. During each timeslice, Gen1 selects a search region and visits t search nodes from it.
3) Restricted backtracking [13] is used. The child tuples of a given tuple in the search tree are ordered according to an evaluation function. Roughly speaking, the best w children are kept, and the others are discarded. Here w can be interpreted as a search width.

Some details are as follows.

Two tuples are *isomorphic* if they are translates of each other. The tuples in a given equivalence class Γ satisfy the same (p, q) pairs:

For all (p, q) and for all tuples $T, U \in \Gamma$,
T satisfies (p, q) iff U does. (4)

Recall that the generators draw tuples from a rectangular pixel array Π. Given a tuple equivalence class Γ, we take the *canonical* tuple of Γ to be the unique tuple in Γ that has been shifted as far left and as far up as possible while remaining wholly in Π. Like Gen0, Gen1's underlying search tree is a tree of tuples, but Gen1's tree contains only canonical tuples. This reduces the search space, but when taken alone this gives no speed-up except perhaps when n is small, by the discussion at the end of Section 4.2. The primary benefit of using only canonical tuples comes from favorable interaction with simulated parallelism and restricted backtracking. This interaction is discussed later in this section.

In Gen1, the two canonical 1-tuples are ignored, and in the search tree, the canonical 2-tuples are the children of the empty tuple. Each search region is a subtree rooted at a canonical 2-tuple. The search proceeds using simulated parallelism as follows. (For simplicity the pseudocode in this section omits some straightforward details, such as what happens when a solution is reached, a tuple fails a constraint, or a search region is exhausted.)

```
// Carry out the search, for given p, q.
procedure search (integer p, q)
    array A initialized to order-children (empty-tuple)
    integer i initialized to 0
    while (No solution has been found) do
        Let R denote the search region A[i].
        // (We suspend this timeslice for R
        // after t nodes are processed.)
        if (R has not been visited before)
            process-node (R)
        else
            Resume prior process-node call for R.
        i ← (i + 1) mod w
```

The *process-node* and *order-children* procedures are now examined in the context of restricted backtracking. The code below shows what happens at each search node.

```
procedure process-node (tuple T)
    array A initialized to order-children (T)
    integer i
    // Search at most w children of T.
    for (i ← 1 to min(w, |A|)) do
        process-node (A[i])

procedure order-children (T)
    array A initialized to the children
        of T in the canonical tuple tree
    integer i
    Sort A in order of best evaluation to worst.
        // (After sorting, A[1] is best.)
    for (i ← 1 to |A| − 1) do
        Swap A[i] and A[i + 1] with probability 5.
    for (i ← |A| downto 2) do
        Swap A[i] and A[i − 1] with probability 5.
    return A
```

Here the children of each search node are first sorted according to an evaluation function. Given the NP-completeness of the tuple-finding problem, it is reasonable to use a heuristic evaluation that may miss solutions. After sorting, the order is perturbed slightly, as discussed below. (Restricted backtracking in general need not include perturbation.) Finally, the best w children are retained and the others are discarded. Backtracking is applied to the subtree of each retained child, in an order determined by the sort order and the perturbation. Gen1 monitors the number of nodes visited during the current timeslice. When this number reaches t, the timeslice ends.

The evaluation of a tuple T is based on the following function:

$$h(T) = \frac{\min_{E \in E^+} p(T, E)}{\max_{E \in E^-} r(T, E)}. \quad (5)$$

Here $p(T, E)$ is the value p_T with respect to the positive exemplar E. The function $r(T, E)$ gives the number of shifts for which exactly $\min(|T|, q)$ pixels of T fit the negative exemplar E. A tuple T_1 is considered to be more desirable than a tuple T_2 if $h(T_1) > h(T_2)$.

The evaluation function $h(T)$ was designed to take both the p and q constraints into account while facilitating fast evaluation with word-parallel bit operations. The complete evaluation uses $h(T)$ along with some computations to break ties and avoid division by 0.

By considering a brute force computation of $h(T)$, it is easy to see that for uniformly-sized characters the time to compute $h(T)$ for a given tuple T is $O(|E^+| + |E^-|)$. Thus if the search width, timeslice, and maximum node limit are held constant, then the search time is, on an idealized computing device, bounded above by a linear function of the number of exemplars. This bounds the running time for all cases by the worst case running time—the time used when the search fails. A precise theoretical analysis of successful searches in this NP-complete problem is probably difficult.

The evaluation function imposes a sort order on the children of a given node. Gen1 slightly perturbs this order, using random numbers. The purpose of the perturbation is to cause different solutions to be found on successive executions. Possibly, this perturbation interacts with simulated parallelism, further contributing to the generation of different solutions on successive executions. One way to generate different solutions on successive executions is to do a full random permutation of the list elements. Since that would destroy all information from the evaluation function, we instead perturb the list only slightly. This slight perturbation preserves much of the information from the evaluation function, and we have found empirically that this perturbation is sufficient to generate different solutions on successive executions. It can be shown analytically that under this perturbation, a given list element usually stays within a few slots of its original position.

Two benefits accrue from searching only canonical tuples rather than all tuples. First, the number of search regions (2-tuples) decreases. Also decreasing, therefore, is the number of 2-tuples that are evaluated during the determination of the best w 2-tuples. When Π is 16 by 16, for example, there are 130,560 2-tuples and 1,920 of these are canonical, so the number of 2-tuple evaluations decreases by a factor of 68. Second, suppose the search regions corresponded to all 2-tuples rather than only the canonical ones. Equivalent tuples have identical evaluations. So, after the 2-

tuples were sorted, most of the w best tuples would be isomorphic, making simulated parallelism a waste.

Related to the tuple generation problem is the question of how many non-isomorphic tuples of a given order n can be formed within a given rectangular pixel array. This counting problem, some variants including generalization to arbitrary dimensionality and dimension sizes, and related combinatorial identities are examined in [18].

4.4 N-Tuple Generation Experiments

Here we summarize the tuple generation experiments. More information may be found in [15].

We ran experiments with Gen0 and Gen1 for n = 4, 7, 10, using the following dichotomies: $c - e$, $e - c$, $e_5 - c_5$, *acenou* – *sxz*, $c - n$. We used Times-Roman eight-point characters digitized at 300 dots per inch. The characters were scanned from laser-printed originals, except for the $e_5 - c_5$ characters; these were scanned from fifth-generation photocopies of laser-printed originals. The experiments were carried out on SPARC 20 computers. The generator parameters were selected so that the generators executed for about five minutes on a particular problem, when no solution was found.

Each of 30 tables produced in the experiments results from one generator executing for a given value of n with a given dichotomy. The experiments represent more than 40,000 invocations of the generators.

The experiments support the following conclusions.

1) Distinguishing tuples can be found reasonably quickly when they exist. The simplest problem is when $(p, q) = (1, n - 1)$; both Gen0 and Gen1 solve such problems in less than 1 second on a SPARC 20.
2) Gen1 is a reasonably efficient algorithm for generating tuples. This becomes apparent when we compare the experiment performance of Gen1 to Gen0's.
3) The execution time of Gen1 is governed by the difficulty of the problem as defined by p and q. With more computing resources, solutions can be found for higher values of p and lower values of q.

The general features of the 30 tables in [15] can be seen by examining Table 1 and Table 2. Table 1 shows the time to find distinguishing 10-tuples for the given (p, q) values, for the $c - n$ dichotomy. Table 2 is for the $e_5 - c_5$ dichotomy. In both tables, the times are averages of 50 searches, and are normalized with respect to the time taken for the ground case of $p = 1, q = n - 1$. Times are shown for the (p, q) values for which all 50 searches found a solution.

The tables show the trade-off between execution time and the difficulty of the (p, q) pair. As we move away from $p = 1, q = n - 1$, solution time increases, until ultimately no solutions are found.

Both tables exhibit a small region of the (p, q) space near $p = 1, q = 9$ where both Gen0 and Gen1 find solutions, and Gen0 finds them faster. In a large area away from $p = 1, q = 9$, where more desirable tuples reside, Gen1 finds solutions and Gen0 does not.

A likely reason that Gen0 finds solutions for the less desirable (p, q) values faster than Gen1 is that Gen1 evaluates many tuples at each search node, even when the search goes directly to a solution with no backtracking. On the other hand, Gen0 evaluates only one tuple at each search node. This leads to an idea for a future hybrid algorithm that runs Gen0 and Gen1 in parallel. If t_0 and t_1 are the respective execution times for Gen0 and Gen1 on a given problem instance, then the hybrid algorithm takes time that is at worst roughly $2\min(t_0, t_1)$. This is in some ways more desirable than the execution characteristics of either generator alone.

Another opportunity for speed-up lies in Gen1's evaluation function. We have found that Gen1's performance is sensitive to changes in the evaluation function.

TABLE 1
NORMALIZED TIME TO SOLUTION FOR THE $c-n$ DICHOTOMY, $n = 10$

Gen1				
	q			
p	6	7	8	9
1	1.9	1.1	.97	1
2	10	1.1	1.0	.94
3	16	5.5	.99	1.0
4		8.1	1.2	1.1
5		16	13	1.0
6		19	17	0.9
7			14	.92
8			17	.88
9				.94
10				.91
15				2.2
20				23

Time for $p = 1, q = 9$: 8262 s

Gen0				
	q			
p	6	7	8	9
1			1.0	1
2			1.3	.95
3				1.0
4				1.1
5				
6				
7				
8				
9				
10				
15				
20				

Time for $p = 1, q = 9$: .1706 s

5 Character Classification Experiments

In this section, we show empirically that n-tuples have several characteristics that are desirable for character classification. Section 5.1 shows that the class-conditional fit probabilities can be usefully controlled by the design parameter n. In Section 5.2, we examine the control exerted by (p, q). Section 5.3 shows that it is possible to find tuples with low mutual correlation. The $e - c$ dichotomy is used as the predominant dichotomy in Sections 5.1 through 5.3, because e and c are difficult to distinguish [14]. Section 5.4 explores class-conditional probabilities for other dichotomies.

The experiments in Sections 5.1-5.4 used second-generation photocopies of Times-Roman eight-point characters that were digitized at 300 dots per inch. Photocopies were used because they vary more than original scanned

TABLE 2
NORMALIZED TIME TO SOLUTION FOR THE $e_5 - c_5$ DICHOTOMY, $n = 10$

Gen1					
	q				
p	5	6	7	8	9
1	2.6	2.6	2.7	1.1	1
2	2.0	2.4	2.3	2.2	1.0
3	3.7	1.7	2.0	2.0	.94
4	6.8	2.2	2.3	1.5	1.4
5	11	3.5	3.2	1.9	.77
6	10	1.7	1.7	1.5	1.5
7	13	3.8	3.7	1.4	1.4
8	28	3.5	2.7	1.4	1.4
9		3.2	2.8	1.3	1.4
10		4.6	4.1	1.3	1.4
15		13	3.5	1.5	1.1
20			11	2.8	.96
25					1.7

Time for $p = 1, q = 9$: 8522 s

Gen0					
	q				
p	5	6	7	8	9
1		28	24	1.0	1
2			22	2.9	1.0
3				23	1.1
4					1.1
5					
6					
7					
8					
9					
10					
15					
20					
25					

Time for $p = 1, q = 9$: .2212 s

characters. Fig. 4 shows some typical characters that were used in the experiments. The characters in the figure were selected from a larger character set using a pseudo-random number generator.

Fig. 4. Some typical characters that were used in the classification experiments.

Eight-point characters are at the limit of type size that can be recognized. Larger type sizes are easier to classify. At 300 dots per inch, n-tuples can distinguish larger characters with virtually no error [11].

5.1 Value of n

Here, we show empirically that Propositions P1 and P2 hold. That is, $P(x_i \mid C^+)$ decreases as n increases, and $P(\overline{x_i} \mid C^-)$ increases with n.

In this experiment, distinguishing tuples for the $e - c$ dichotomy were generated for $n = 3, 4, \ldots, 15$. A training set comprised 10 es and 10 cs. A trial for a given value of n consisted of three steps. First, a dichotomy was formed by randomly selecting one e exemplar and one c exemplar. Next, an n-tuple was generated for the dichotomy with $p = 1$ and $q = n - 1$. Then the class-conditional probabilities were estimated by averaging over a test set of 900 es and cs.

The results for each value of n were averaged over 50 trials. The same test set was used for all trials.

Fig. 5 shows the empirical probabilities of correct classification as a function of n. Propositions P1 and P2 hold.

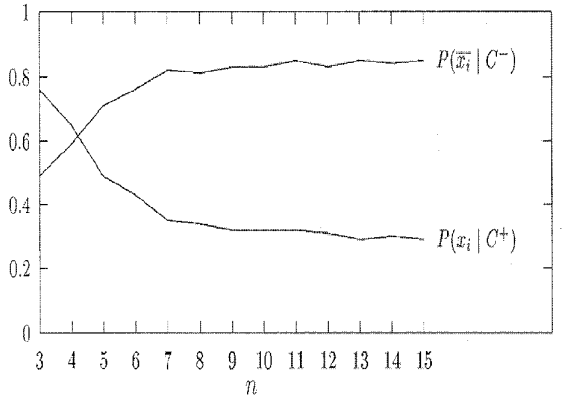


Fig. 5. Class-conditional probabilities as a function of n.

It follows that there exists some value of n for which the following holds approximately:

$$P(x_i \mid C^+) = P(\overline{x_i} \mid C^-). \tag{6}$$

In the absence of prior constraints such as information about the class distribution, it is reasonable to equalize the probabilities of correct classification for the positive and negative classes, as in (6). Fig. 5 suggests that here we want $n = 4$ or $n = 5$.

It is fortunate that relatively small values of n such as 4 or 5 are good, because usually as n decreases, tuple generation becomes easier.

5.2 Relation Between (p, q) and Class-Conditional Fit Probability

In this section, we show that for a given value of n, the parameter p affects primarily the positive class-conditional probabilities, and q mainly the negative class-conditional

probabilities. This experiment was conducted as in the previous section, but with different values of n, p, and q: here we used n = 3, 7, 10, p = 1, 3, 5, and selected values of q. Table 3 shows the results.

TABLE 3
CLASS-CONDITIONAL FIT PROBABILITY FOR THE $e-c$ DICHOTOMY, AS A FUNCTION OF n, p, AND q

n = 10						
	q = 8		q = 7		q = 6	
p	$P(x_i \mid e)$	$P(\overline{x_i} \mid c)$	$P(x_i \mid e)$	$P(\overline{x_i} \mid c)$	$P(x_i \mid e)$	$P(\overline{x_i} \mid c)$
1	.143	.961	.093	.977	.200	.994
3	.777	.904	.795	.995	.767	.997
5	.942	.965	.949	.994	.945	.996

n = 7						
	q = 6		q = 5		q = 4	
p	$P(x_i \mid e)$	$P(\overline{x_i} \mid c)$	$P(x_i \mid e)$	$P(\overline{x_i} \mid c)$	$P(x_i \mid e)$	$P(\overline{x_i} \mid c)$
1	.349	.821	.206	.925	.240	.987
3	.840	.554	.841	.917	.814	.993
5	.949	.660	.941	.947	.944	.994

n = 3		
	q = 2	
p	$P(x_i \mid e)$	$P(\overline{x_i} \mid c)$
1	.762	.492
3	.902	.426
5	.981	.500

Increasing p while leaving q unchanged improves $P(x_i \mid e)$ with little or no effect on $P(\overline{x_i} \mid c)$. Similarly, decreasing q while leaving p unchanged improves $P(\overline{x_i} \mid c)$ with little or no effect on $P(x_i \mid e)$. The results confirm that we should use high p and low q.

Table 3 also shows that by selecting different values of (n, p, q), we can move the probabilities closer to the optimal values for either the positive or the negative class. This is useful for classification tasks where the a priori probabilities of the two classes are unequal, as is usually the case in natural language.

The table suggests that given two tuples x_i and x_j of the same order, if x_i dominates x_j as in Section 2, then x_i is a better classifier than x_j, i.e.,

$$P(x_i \mid C^+) \geq P(x_j \mid C^+) \text{ and } P(\overline{x_i} \mid C^-) \geq P(\overline{x_j} \mid C^-),$$

where one of the inequalities holds strictly. For instance, the tuples generated with (n, p, q) = (7, 5, 4) are, on the average, better classifiers than those generated with (7, 1, 6).

5.3 Statistical Correlation

Suppose we have a collection of classification features and their class conditional probabilities for a given dichotomy. If the features are independent, then their discriminating abilities can be usefully combined. On the other hand, if the features are strongly correlated with each other, then using several features yields little more information than using one feature alone. Generally, it is desirable for classification features to be independent rather than correlated [3]. In this section we show that it is possible to find tuples with low mutual correlation as classification features.

The experiment was conducted using one e and one c to form an instance of $e-c$ dichotomy. We used n = 7 and repeated the experiment with different (p, q) constraints: (0, 7), (1, 6), (5, 5), and (7, 4). Tuples were generated using Gen1, except in the unconstrained case (p = 0, q = 7), where tuples were generated by selecting seven distinct random pixel locations from the positive character's frame. A fixed test pool of 900 es and cs was used. For each (p, q) value used, the experiment consisted of 100 trials. Each trial went as follows.

Two new tuples were generated that satisfy the specified (p, q) constraint. Each tuple was tested on the 900 es. From this we constructed a contingency table tallying the occurrences of the possible outcomes (success-success, success-failure, failure-success, failure-failure). An estimate of the correlation coefficient was computed from the contingency table, unless either tuple matched all 900 es. When this occurred it was impossible to meaningfully estimate the correlation coefficient; such occurrences were recorded. Finally, a trial was completed by repeating the estimation procedure for the same two tuples on the 900 cs.

Table 4 shows a (reverse) cumulative histogram of the squares of the correlation coefficients. The first row shows how many of the 100 tuple pairs had correlation coefficients defined. The pairs with undefined correlation coefficients had one or both tuples behaving perfectly on all 900 entries in the test set.

TABLE 4
DISTRIBUTION OF SQUARES OF CORRELATION COEFFICIENTS OF 100 TUPLE PAIRS GENERATED FOR THE $e-c$ DICHOTOMY, n = 7

	p = 0, q = 7 Random		p = 1, q = 6 Gen1		p = 5, q = 5 Gen1		p = 7, q = 4 Gen1	
	e	c	e	c	e	c	e	c
Defined	27	40	99	90	58	87	43	97
>.00	27	40	99	90	58	87	43	97
>.05	3	3	21	18	18	53	17	94
>.10	2	2	9	7	10	37	9	86
>.15	1	2	4	6	7	31	8	72
>.20	1	1	2	4	6	22	7	59
>.25	1	0	2	3	5	14	7	49
>.30	1	0	2	2	4	12	7	44
>.35	0	0	2	2	3	10	5	40
>.40	0	0	1	2	3	9	4	31
>.45	0	0	1	2	3	8	4	26
>.50	0	0	1	1	3	5	3	12
>.55	0	0	1	1	3	4	3	12
>.60	0	0	1	1	3	4	3	7
>.65	0	0	1	1	3	4	3	7
>.70	0	0	1	1	1	4	1	4
>.75	0	0	1	1	0	4	1	3
>.80	0	0	1	1	0	4	1	3
>.85	0	0	1	1	0	4	1	1
>.90	0	0	1	1	0	4	1	0
>.95	0	0	1	1	0	4	1	0

Two observations emerge from the table. First, it is possible to generate tuples that have low correlation. For example, over 90% of the tuples for p = 1, q = 6 have a squared correlation coefficient of .10 or less. Second, as we move away from p = 1, $q = n - 1$ in the $p - q$ space, correlation increases. There is a trade-off. Feature effectiveness increases (Section 5.2), but so does correlation.

TABLE 5
CLASS-CONDITIONAL PROBABILITIES FOR SELECTED DICHOTOMIES

Dichotomy $C^+ - C^-$	Fit Frequency For C^+			Miss Frequency For C^-			Standard. Deviation Of C^+ Fit Freq.			Standard Deviation Of C^- Miss Freq.		
$e - c$			.927			.948			.120			.136
$c - e$			.942			.948			.052			.088
$o - x$			.998			.998			.005			.007
$x - o$			.965			.990			.093			.026
$no - ce$		.950	.972		.977	.941		.140	.091		.043	.094
$ce - no$		.947	.974		.919	.949		.083	.054		.092	.068
$ceo - sxz$	.998	.988	.999	.999	.998	.985	.005	.023	.008	.002	.005	.045
$sxz - ceo$	.936	.978	.996	.973	.951	.937	.139	.054	.014	.040	.058	.071

5.4 Choice of Dichotomy

So far we have only examined the $e - c$ dichotomy. Table 5 shows empirical class-conditional probabilities for each class for selected other dichotomies. This experiment's design is the same as that in Section 5.1, except here (n, p, q) is taken to be (6, 4, 4). The table shows the average fit frequency among 50 tuples for the positive and the negative classes, and the standard deviation.

One observation emerging from this table is that the direction of the dichotomy matters; there is an asymmetry between the positive and negative classes. For example, tuples for *ceo-sxz* were found to have better performance than the tuples for *sxz-ceo*.

6 DISCUSSION AND CONCLUSIONS

Features for character recognition that have been repeatedly explored in the last forty years include masks and templates; moments and moment-invariants; principal components; orthogonal decompositions (Fourier, Hadamard, Haar, and Walsh transforms); contours; lakes, bays, and lids; stroke crossings, curvatures, and end-points; projections; and dynamic contour warping. All of these features suffer from one of two defects: either they must be constructed *by hand* according to the designer's experience with the character shapes under consideration, or they are based on the shape of the patterns in each class, rather than on the *difference* in shape between classes.

Although implicitly any classifier is based on the difference between classes, few OCR classifiers work directly on the raw bitmaps. This is because of two problems with bitmaps: they are large, and shift-variant. Invariably, classifiers solve these problems by extracting features from the bitmap, as in neural networks and statistical classifiers. The input to such classifiers generally consists of a fixed set of features, often pixels averaged over a small region. Unlike n-tuples, such features do not adapt to the sometimes subtle pixel arrangements that differentiate the classes.

In principle, classifiers based on pairwise bitmap discriminants can be implemented, but in practice this has not been done, except for decision trees. Decision trees based on bitmaps cannot cope with shift invariance. Decision trees not based on bitmaps must use some kind of features. Our n-tuple features are intended to be such a feature.

Although tuple features can, and have been, used with any of the standard classifiers based on binary feature vectors, they are particularly appropriate for binary decision trees. In such a tree each node is responsible for a specific dichotomy. The relationship that we have demonstrated between the tuple design parameters and the class-conditional feature probabilities can be exploited to construct statistical decision trees based on a small training sample, and to predict the error rate of such tree classifiers [5].

The tuple generation method that we have described makes no explicit use of features such as those listed earlier. Tuple features are free from the two defects discussed above.Tuple features are generated automatically and are independent of the particular nature of the symbols, and they are based on the difference between designated classes.

The generation method can be applied to any set of isolated printed symbols, such as those in non-Roman alphabets and ideographs. In fact the approach is intended to be sufficiently general that it can be applied to textures, fingerprints, manufacturing defects, and notation from bridge, chess, and music. The generation of tuple features is completely automatic, and can be carried out in a few seconds on contemporary workstations.

Furthermore, the method fosters the statistical independence between multiple features for the same dichotomy, which greatly facilitates classifier design and error estimation. Therefore the technology renders possible, in principle at least, the completely automated design of features and classifiers on the basis of a small labeled design sample. Such a completely automated design procedure is essential for any adaptive or learning scheme that requires modification of both features and classifiers.

ACKNOWLEDGMENTS

We would like to thank Robin Flatland, Edward Green, and Prateek Sarkar for their comments about a draft of this paper. Mr. Sarkar also assisted with the statistical correlation experiments. We also thank the anonymous referees for their thorough reviews. We gratefully acknowledge the support of the Central Research Laboratory, Hitachi, Ltd., of the Educational and Research Networking of Northern Telecom—BNR, Inc., and of the New York State Center for Advanced Technology (CAT) in Automation, Robotics and Manufacturing at Rensselaer Polytechnic Institute. The CAT is partially funded by a block grant from the New York State Science and Technology Foundation. We are grateful to Dr. Hiromichi Fujisawa for suggesting, in 1990, that we take a fresh look at n-tuples.

REFERENCES

[1] R. Bakis, N.M. Herbst, and G. Nagy, "An Experimental Study of Machine Recognition of Hand-Printed Numerals," *IEEE Trans. Systems, Science, and Cybernetics*, vol. 4, no. 2, pp. 119-132, July 1968.

[2] W.W. Bledsoe and I. Browning, "Pattern Recognition and Reading by Machine," *Proc. Eastern Joint Computer Conf.*, no. 16, pp. 225-233, Dec. 1959.

[3] R.O. Duda and P.E. Hart, *Pattern Classification and Scene Analysis*, pp. 32-33. John Wiley & Sons, 1973.

[4] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, 1979.

[5] D.M. Jung, "Joint Feature and Classifier Design for OCR Based on a Small Training Set," PhD thesis, Rensselaer Polytechnic Inst., May 1995.

[6] L.A. Kamentsky and C.N. Liu, "Computer-Automated Design of Multifont Print Recognition Logic," *IBM J. Research and Development*, vol. 7, no. 1, pp. 2-13, 1963.

[7] L.A. Kamentsky and C.N. Liu, "A Theoretical and Experimental Study of a Model for Pattern Recognition," *Computer and Information Sciences*, pp. 194-218. Washington, D.C.: Spartan, 1964.

[8] M.D. Levine, "Feature Extraction: A Survey," *Proc. IEEE*, vol. 57, no. 8, pp. 1,391-1,407, Aug. 1969.

[9] C.N. Liu and G.L. Shelton, "An Experimental Investigation of a Mixed Font Print Recognition System," *IEEE Trans. Electronic Computers*, vol. 15, no. 6, pp. 916-925, Dec. 1966.

[10] M. Nadler, "The State of the Art in Optical Character Recognition," *Machine Perception of Patterns and Pictures*, pp. 3-18. Teddington, Middlesex: Inst. of Physics, 1972.

[11] G. Nagy and X. Wang, "Automatically-Generated High-Reliability Features for Dichotomies of Printed Characters." *Proc. Symp. Document Analysis and Information Retrieval*, Las Vegas, Apr. 1996.

[12] V.N. Rao and V. Kumar, "On the Efficiency of Parallel Backtracking," *IEEE Trans. Parallel and Distributed Systems*, vol. 4, no. 4, pp. 427-436, Apr. 1993.

[13] N.J. Nilsson, *Principles of Artificial Intelligence*, p. 72. Tioga Publishing, 1980.

[14] S.V. Rice, J. Kanai, and T. Nartker, "The Third Annual Test of OCR Accuracy," Ann. Report, UNLV Information Science Research Inst., p. 15, 1994.

[15] A. Shapira, "Experiments on the Generation of Distinguishing N-tuples for Selected Character Dichotomies," Rensselaer Polytechnic Inst., ECSE Dept., Technical Report no. ECSE-OCR-20DEC95, Dec. 1995.

[16] A. Shapira, Home Page, URL http://www.cs.rpi.edu/~shapiraa/, Internet World Wide Web, 1996.

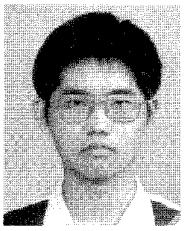
[17] A. Shapira, PhD thesis. Rensselaer Polytechnic Inst., 1996 (in preparation).

[18] A. Shapira and M. Krishnamoorthy, "Enumeration of the Patterns in a Lattice," Rensselaer Polytechnic Inst. Computer Science Technical Report no. 95-9, May 1995, submitted for publication.

[19] F.W.M. Stentiford, "Automatic Feature Design for Optical Character Recognition Using an Evolutionary Search Procedure," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 7, no. 3, pp. 349-355, May 1985.

[20] J. Ullman, "Experiments with the n-Tuple Method of Pattern Recognition," *IEEE Trans. Computers*, vol. 18, no. 12, pp. 1,135-1,137, Dec. 1969.

[21] X. Wang, "Improving N-Tuples and Dichotomizers in N-Tuples Based Decision Tree Classifier," master's thesis, Rensselaer Polytechnic Inst., Dec. 1995.

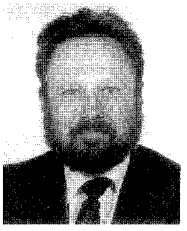


Dz-Mou Jung received the BS degree in electrical engineering from National Taiwan University in 1985, and the MS and PhD degrees in computer and systems engineering from Rensselaer Polytechnic Institute in 1989 and 1995, respectively.

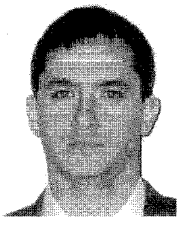
He spent 1991 at the IBM T.J. Watson Research Center, Yorktown Heights, New York, conducting multimedia research. From March 1995 to December 1995, he was a consultant with GE R&D Center. Dr. Jung is currently a software engineer with Caere Corporation, developing document analysis and optical character recognition systems. His research interests include document image understanding, multimedia, image processing, and computational geometry. He is a member of the IEEE Computer Society.

M.S. Krishnamoorthy received the BE degree (with honors) from Madras University in 1969, the MTech degree in electrical engineering from the Indian Institute of Technology, Kanpur, in 1971, and the PhD degree in computer science, also from the Indian Institute of Technology, in 1976.

From 1976 to 1979, Dr. Krishnamoorthy was an assistant professor of computer science at the Indian Institute of Technology, Kanpur. He joined Rensselaer Polytechnic Institute in 1979 as an assistant professor of computer science, and was promoted to associate professor in 1985. His research interests include the design and analysis of combinatorial and algebraic algorithms and programming environments.



George Nagy received the BEng and MEng degrees from McGill University, and the PhD degree in electrical engineering from Cornell University in 1962 (on neural networks). For the next 10 years he conducted research on various aspects of pattern recognition and OCR at the IBM T.J. Watson Research Center, Yorktown Heights, New York. From 1972 to 1985, he was a professor of computer science at the University of Nebraska, Lincoln, and worked on remote sensing applications, geographic information systems, computational geometry, and human-computer interfaces. Since 1985, he has been a professor of computer engineering at Rensselaer Polytechnic Institute. He has held visiting appointments at the Stanford Research Institute, Cornell University, the University of Montreal, the National Scientific Research Institute of Quebec, the University of Genoa and the Italian National Research Council in Naples and Genoa, AT&T Bell Laboratories, IBM Almaden Research Center, McGill University, and the Institute for Information Science Research at the University of Nevada. In addition to document image analysis and character recognition, Dr. Nagy's research interests include solid modeling, finite-precision spatial computation, and computer vision.



Andrew Shapira received the BS degree in computer science (engineering) from the University of Illinois at Urbana-Champaign in 1985. From 1985 to 1987, he was a software development engineer at Convex Computer Corporation in Dallas, Texas, where he worked on Convex's verson of Unix for their supercomputers. In 1990, Mr. Shapira received the MS degree in computer and systems engineering at Rensselaer Polytechnic Institute (PhD expected in 1996). His research interests include combinatorics, graph theory, search problems, backtracking, and the construction of practical software tools.

Mr. Shapira is the originator of Randpack, an ANSI C package for the convenient, portable, and robust generation and testing of pseudorandom numbers and other sequences. In the late 1970s, while in their mid-teens, Bruce Maggs, C. David Sides, and Mr. Shapira created "Avatar," a computer game that has seen roughly 1,000,000 hours of use and remains popular today.