National Library of Canada
Canadian Theses Service
Ottawa, Canada
K1A 0N4

Bibliothèque nationale du Canada
Services des thèses canadiennes

CANADIAN THESES

THÈSES CANADIENNES

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30.

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30.

THIS DISSERTATION HAS BEEN MICROFILMED EXACTLY AS RECEIVED

LA THÈSE A ÉTÉ MICROFILMÉE TELLE QUE NOUS L'AVONS REÇUE

A PC-Based, Multifont, Character Recognition System

Shoshana Goldberg

A Major Report

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements
for the degree of Master of Computer Science at
Concordia University
Montréal, Québec, Canada

March 1987

ISBN 0-315-35510-7

ABSTRACT

A PC-Based, Multifont, Character Recognition System

Shoshana Goldberg

This project describes the design and implementation of a fast, accurate character recognition system for the IBM PC. The system described supports the recognition of Courier, Gothic, Elite and Pica font types. Using a modified form of the crossing algorithm, feature determination within the various fonts was determined by analysis of the prevalent topological information. The success criteria were determined by simple mathematical and spatial groupings. Differences between various recognition speeds are discussed. In addition, the resulting analysis is discussed vis-a-vis the generated confusion tables.

ACKNOWLEDGEMENTS

I would like to express my gratitude to Dr. C.Y. Suen for his guidance and support in the preparation of this project.

I would also like to thank Mike Yu and Mrs. Joyce Pope for their assistance in providing me with the data required for this project.

TABLE OF CONTENTS

1.0 Introduction

The goal of this project was to produce a character recognition system for the PC that would handle type-written characters of four fonts. The fonts to be used were: Gothic, Elite (12 pitch), Courier, and Pica (10 pitch). The major requirements for the system were for fast, inexpensive and accurate recognition of characters.

The field of character recognition has literally exploded onto the scene with the advent of relatively low cost scanners (ranging in price from US$1685 to US$39,000) available for small computer systems which contain OCR (optical character recognition) capabilities. For a complete survey of available scanning systems, see Stanton, Burns & Venit, 1986 [7]. Unfortunately this article does not provide detailed success rates of the various systems and actual recognition speeds. Speeds are merged with scanning speeds and are not provided on a 'per character' basis which would be more meaningful for our purposes.

The need for a means to transform printed (i.e. typewritten) documents into a form that would allow for their storage within computer systems has become more pronounced with the widespread use of computers and computer networks in the workplace and home. The ability to scan such documents and automatically be able to store them in ASCII format would greatly reduce the need for entering such documents.

Text scanners scan documents a line at a time, isolate the characters and store them in RAM for the recognition system to handle. Character recognition is carried out either by simple matrix matching of the character to those recognized by the system, or by the more sophisticated pattern recognition methods. The latter involves the analysis of the character for various features (i.e. contours, closed or open loops, etc). On the basis of these features the characters are identified.

The process of character recognition requires certain basic steps. For a full description of the functions and operations of the various stages in an OCR system, refer to Suen, 1986 [9]. The following is a brief summary in order to familiarize the reader with the issues involved.

A typical OCR system is diagrammed below. It begins with the data (document form) which is passed through the optical scanner. The various steps may be described as follows (complete description may be found in Suen '86 [9]):

Scanning:

The scanner digitizes the document. Typically this involves creating a digital representation of the image that may be passed on for further

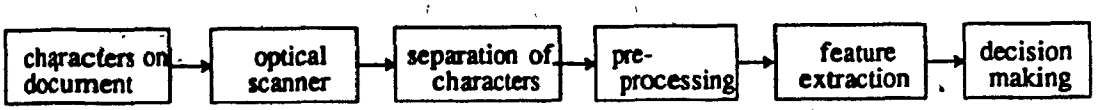


Figure 1. Steps in a typical OCR system

processing or stored immediately for future recovery.

Separation of Characters:

The second step is to isolate each individual character in the document for analysis by the recognition system. In the work described below, this involved a 40x40 ASCII matrix which represented the character. (Examples of the data are included in Appendix B)

Preprocessing

Once the character is separated, it is passed through a preprocessor which smooths the edges of the character, and removes any noise which would inhibit the recognition of the character. Smoothing techniques involve looking at each pixel and setting it to black or white depending on the settings of its neighbors. A simple approach [3] involves setting a pixel to white [black] if all its neighbors are white [black]. A more sophisticated approach is presented in Section 4.1.1.

Skeletonization [4] (i.e. creating a skeleton of the character) may be used as a prepocessing technique which extracts features (see feature extraction below) by eliminating excess information. Other techniques such as this involve the shrinking of the character to half-size. These techniques may speed up processsing later on while maintaining the features of the character.

Another preprocessing technique involves the normalization of the character's size and orientation to that expected by the recognition system. (This is particularly important in the recognition of handprinted characters. Normalization will remove differences caused by the thickness of the pen used or the angle at which the character was written. In processing the typewritten characters below the only normalization required was the positioning of the character within the matrix.)

Feature Extraction

After preprocessing, the character is then ready to be passed through the feature extraction module in which the system analyzes the character and extracts from it the various characteristics that will allow it to be identified. Feature extraction includes methods which analyze the structure of the character such as contour following (where the contours of the character are analyzed[1, 6, 9]) or techniques which yield the postions of straight lines or edges within a character; methods which perform a global analysis of the character - the analysis of geometrical moments [10] or results of transforms (ex: Fourier), physical

measurements of the character (height, width) or density of pixels. From these features a set of primitives which make up the character as a whole may be derived to deduce the identity of the character.

Decision Making

The last step is the actual decision making process which yields the identity of the character. The decision making process typically includes the logic by which a particular character is recognized. This may vary in complexity from a simple best fit template matching algorithm[3] to a highly complex set of rules. [6] In a typical OCR system, this produces the ASCII representation of the character.

A major consideration that must be taken into account in any OCR system which handles typed characters is the font types that will be handled. Different fonts may contain significantly different representations of the same character. A typical example is the lower case 'G'. In Times Roman font the character looks like 'g'. However in Courier it looks like 'g'. It is thus clear that a single set of features cannot be employed for a given character in all fonts. However, within a font, each character can be assumed to be generated with a fair degree of consistency. There is only so much variation that a typewriter can produce in a single character. For such systems it is not necessary to provide complex feature extraction techniques. As described above, matrix matching algorithms or simple topological features are often enough to provide adequate recognition rates.

Such issues become even more pronounced when dealing with handwritten characters. In this case, it is no longer sufficient to use simple techniques of matrix matching of a character against those recognized by the system. A system that deals with handprinted characters is required to handle far more variation in the representation of a given character. For such systems, it is necessary to utilize feature detection algorithms that analyze the formation of the character itself as alluded to above.[2, 8, 9]

2.0 Materials & Methods

2.1.1 Hardware

The initial coding was performed on a a Plexus, P/35, a UNIX-based mini-computer. Once initial coding was completed, the final stages of development were performed on a Philips PC (Single floppy, 10Mbyte hard-disk, IBM compatible). The PC was running under MS-DOS 2.11 and had 512K RAM.

Note: all timing results given below were taken from this PC which is approximately 30% slower than a regular IBM PC.

2.1.2 Software

The software was written in the C programming language [5] and compiled under Lattice C, (Lattice C Compiler, Version 2.12, Lattice, Inc.) using the Microsoft Linker (Version 2.0, Microsoft Inc.).

2.1.3 Data Format

The data was represented in 40x40 arrays of ASCII characters which provided a binary representation of the pixels of the characters. Blank space was taken to be a white pixel (0) with a non-blank space taken to be a black pixel (1).

Input was produced from type-written pages containing the character set to be supported. These sheets were scanned with a Microtek 200 at 200 DPI (dots per inch). The output produced was in encoded form. This was decoded to produce the data in the required format. Samples of the sheets scanned are included in Appendix A.

The four fonts used were: Gothic (12 pitch), Elite (12 pitch), Pica (10 pitch), and Courier (10 pitch).

2.2 Criteria for Selection of an Algorithm

The objective of this project was to produce positive recognition of typed characters on an IBM PC with the maximization of speed as the primary concern.

Thus the criteria used for selecting an algorithm were the following:

- speed on a PC. This precludes any highly CPU intensive operations.
- high recognition rate
- low cost
- ease of implementation in a C environment.

In order to maximize speed and minimize cost, it was important to select an algorithm that produced the maximum amount of information with respect to a character's features while consuming a minimum of processing time.

The crossing method algorithm was selected because it required little computational overhead in extracting the required features from the data. In order to glean the required crossing information it was necessary to pass through the data twice. However no complicated operations were required (as can be seen below in the description of the algorithm). Other features, such as height and width of the characters (as well as number of pixels within a character) are simply garnered as the character is read in, thus not contributing any great costs. Pattern matching was performed on the basis of a best fit algorithm. In addition the results (as seen below) produced relatively high recognition rates (98%+).

It should be noted that the selection of an algorithm did not take into consideration the scanning speed or the time taken to read in the data. It is assumed for the purpose of this project that the data is available to the recognition system on demand much in the same way for the OCR systems as described above where the characters are resident in RAM for access by the recognition system.

2.3. The Crossing Method

The crossing method algorithm is a relatively simple one. It searches for the crossing number for each line of data. For the purpose of this project, crossing numbers were calculated horizontally and vertically. These crossing numbers were stored in an array and compared against a data base of these features specifically created for each font. The decision making process simply had to extract the best match.

The crossing number is calculated from the number of dark blocks of pixels that may be found in a particular line of data. For instance, if the data line being checked contains a single line of dark pixels, then the crossing number will be set to one. A schematic representation of this algorithm can be seen below.

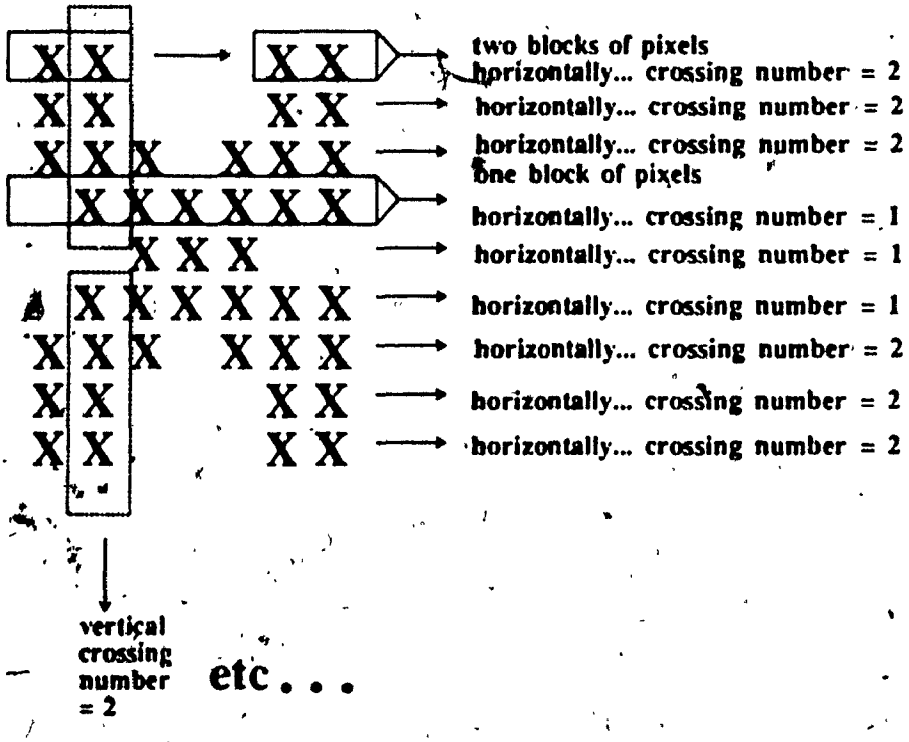
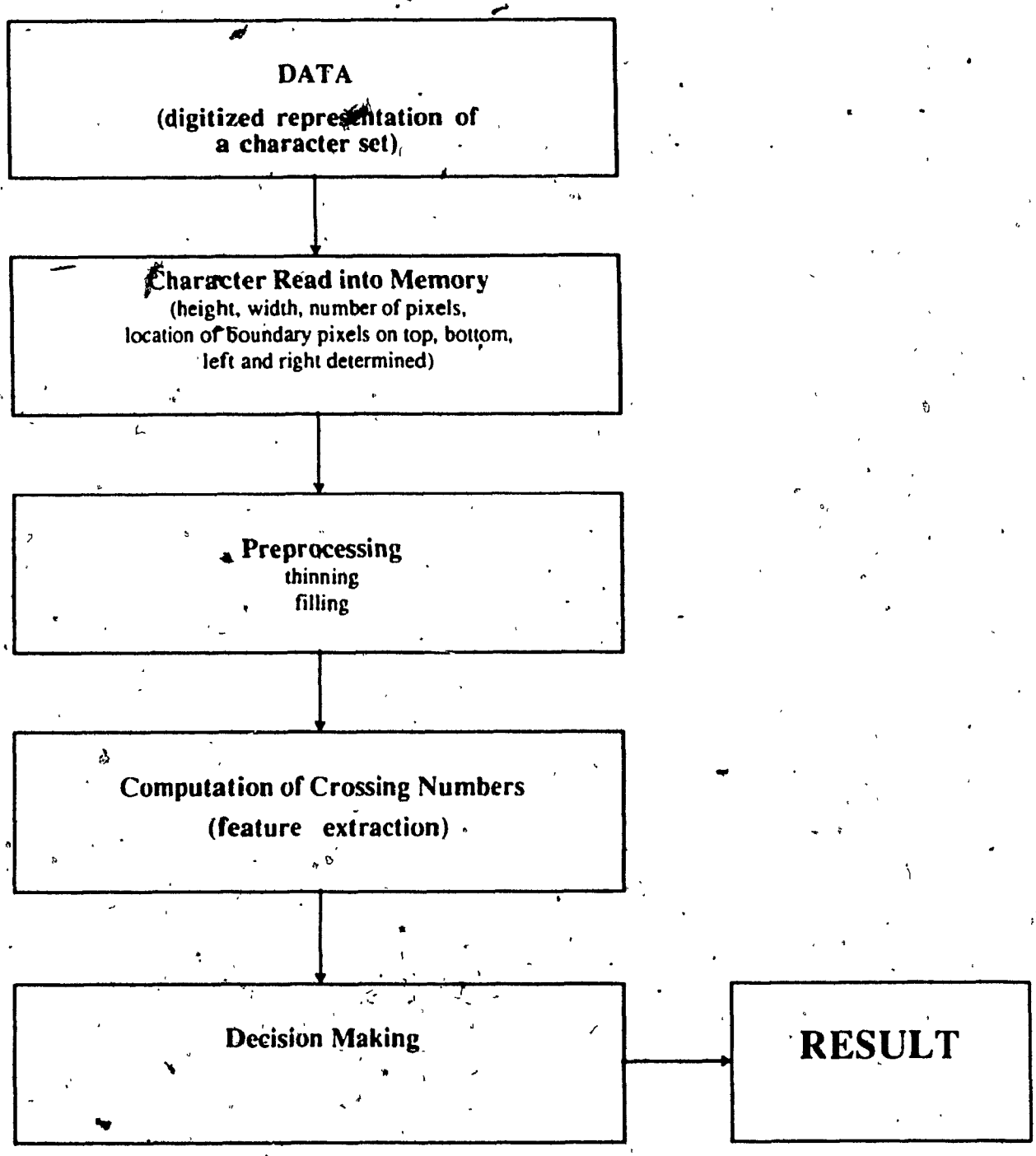


Fig. 1

2.4 Flowchart of Recognition System



2.5 Decision Trees

The following decision trees depict how the system processes the raw data. Physical information was used to speed up processing. This includes:

- the density of the character, calculated from the number of black pixels in the character relative to the total number of pixels (both white and black) (calculated within the actual dimensions of the character)

- the width of the character, especially convenient for isolating very narrow characters such as 'I', 'l', ';', and ':'.

- the height of the character, useful for the determination of the identity of such characters as '-', '_', etc.. as short characters, and '(', ')', and 'j' as typically long characters.

The thresholds for these values were determined by the nature of the given font being handled. In the case of the Gothic font, for example, these thresholds were as follows: (*note: all calculations were done with respect to the height and width of the the character itself within the data matrix, The categories listed below describe those found in the decision tree in Section 2.5.1.1*)

density high -more that 150 black pixels in the character

square -height and width of the character almost equal (range allowable - 2 pixels)

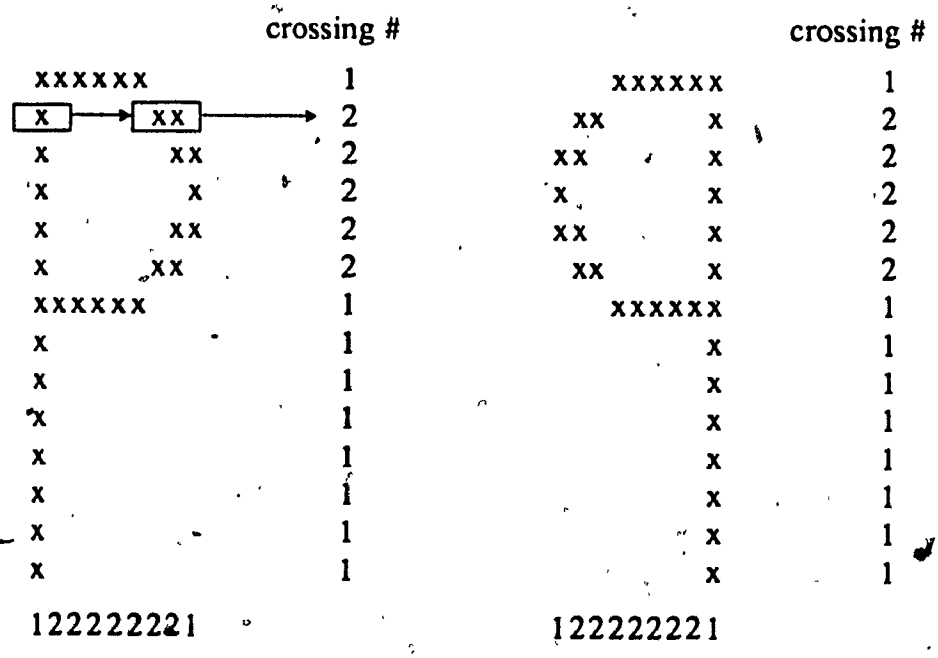


Fig. 2 Examples of ambiguous crossing counts

very short	-maximum heigh of character is 12 pixels.
narrow	-less that 10 pixels wide
lower case	-less that 16 pixels wide and height between 8 and 17 pixels.
very tall	-height greater than 25 pixels.
> 110 pixels	-character contains more than 110 black pixels.
% density	-a given percentage density is calculated by finding the number of black pixels relative to the total number of pixels (black and white) in the character.
rest	-all characters that may fall through the decision tree (i.e. do not fall into any of the categories described above)

The diagrams below should give one a clear idea of the characteristics that separate groups of characters. These are distinctly different for the different fonts used. It should also be noted that certain fonts lend themselves better to this manipulation than others. This may be noted by the number of characters per branch of the tree.

Furthermore, an additional decision tree was required for characters that produce identical crossing numbers and are yet different. An example of this problem occurs with 'p' and 'q' in Gothic. For simplicity a skeletized version of these characters may be seen in Figure 2.

As can be seen in Figure 2, these two characters produce identical crossing numbers. Thus in order to distinguish between them more information is required. In this case, this may be accomplished by making a simple count of the number of black pixels in the first two columns of the character. If the number is over a given threshold the character may be considered to be a 'p', otherwise it is considered a 'q'. This flag is also useful in reducing the number of characters within a branch of the tree. (i.e. it may also separate 'J' from 'L', etc).

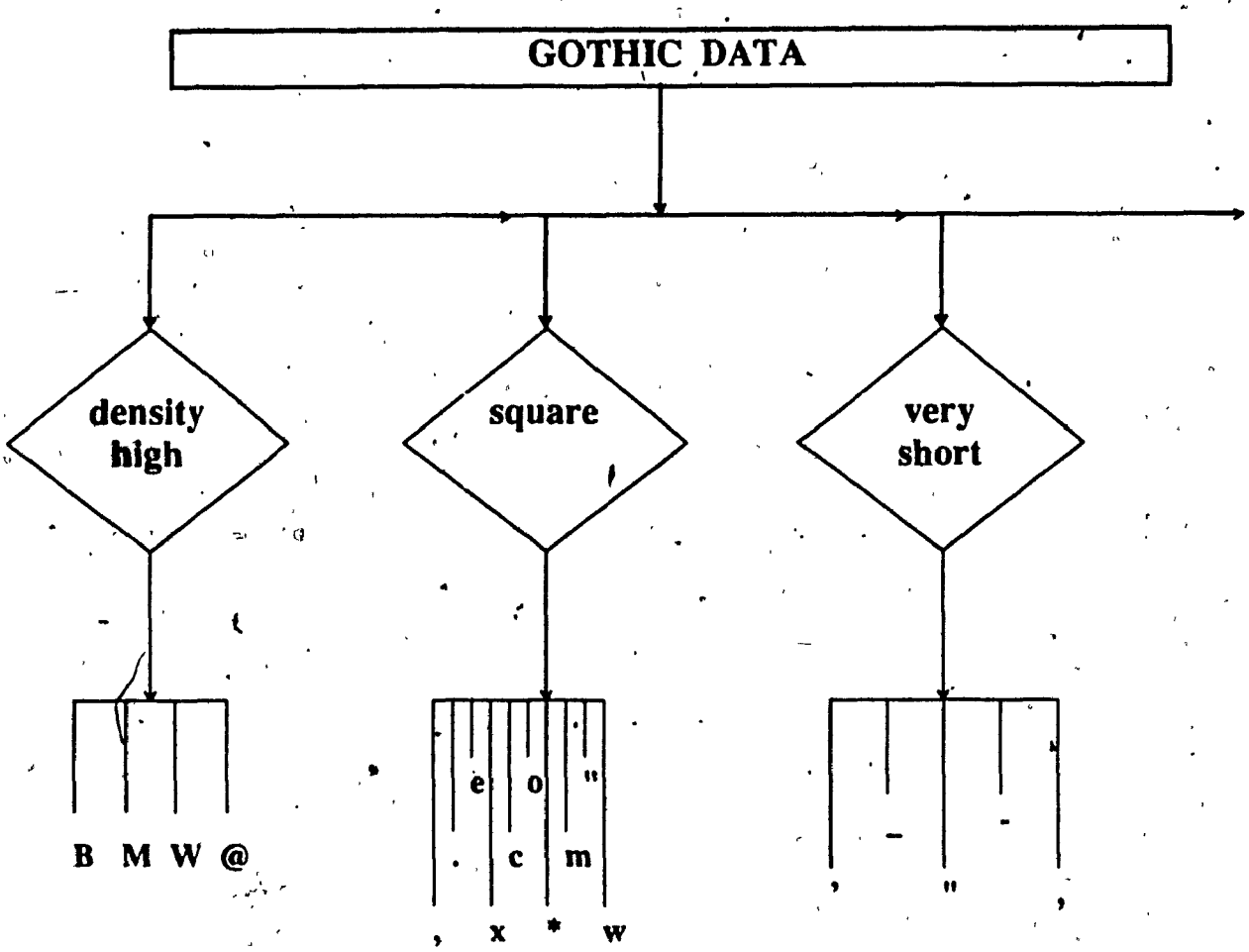
(This flag is referred to in the trees below simply as **flag.** It is used to break up particularly dense branches. (If flag is true then the characters with a vertical line in the first few columns are checked for,, ex. 'p', 'd'))

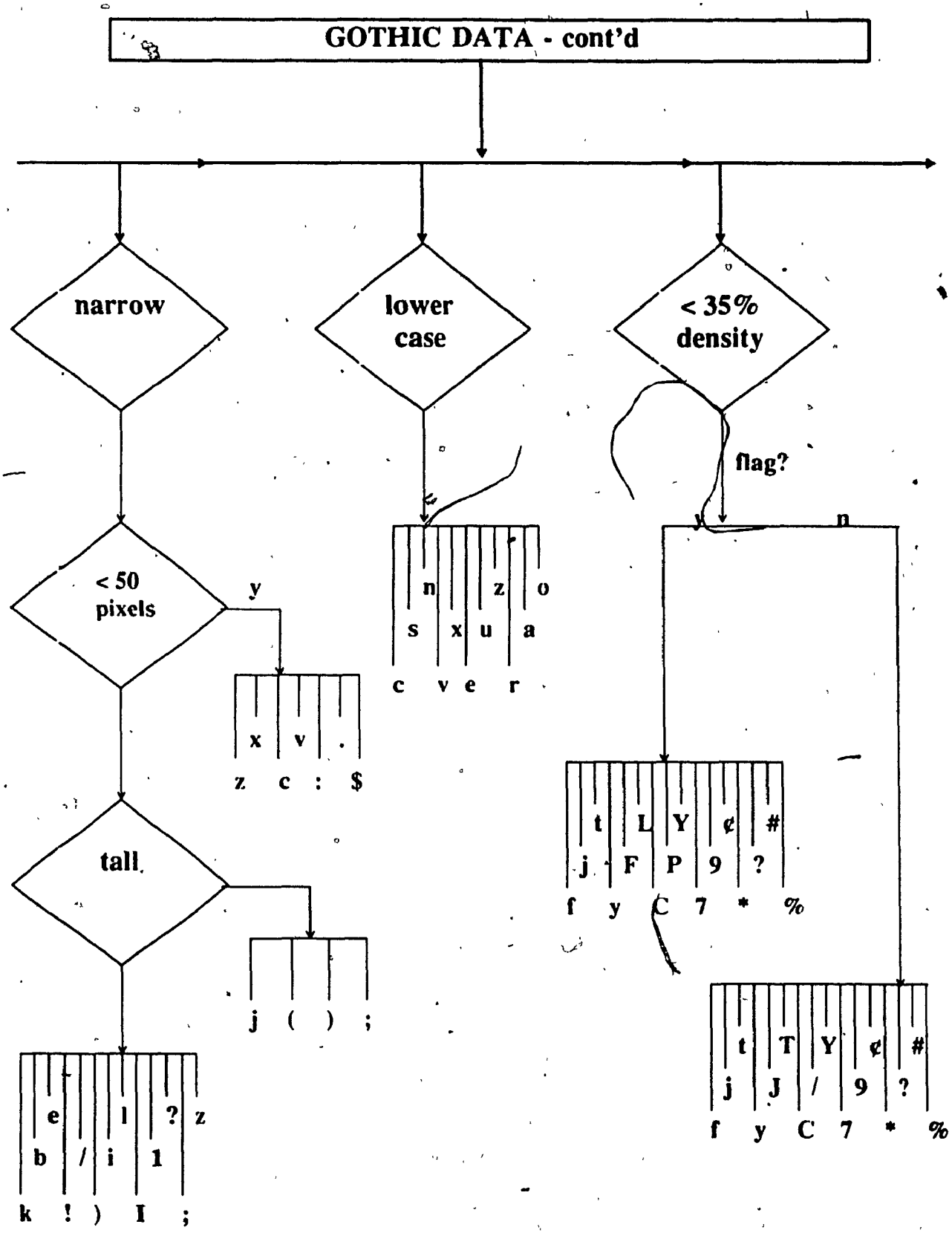
Other calculations are also used to distinguish between similar characters. For instance, the number of pixels in the top left hand corner of a character helps distinguish between a 'D' and an 'O'.

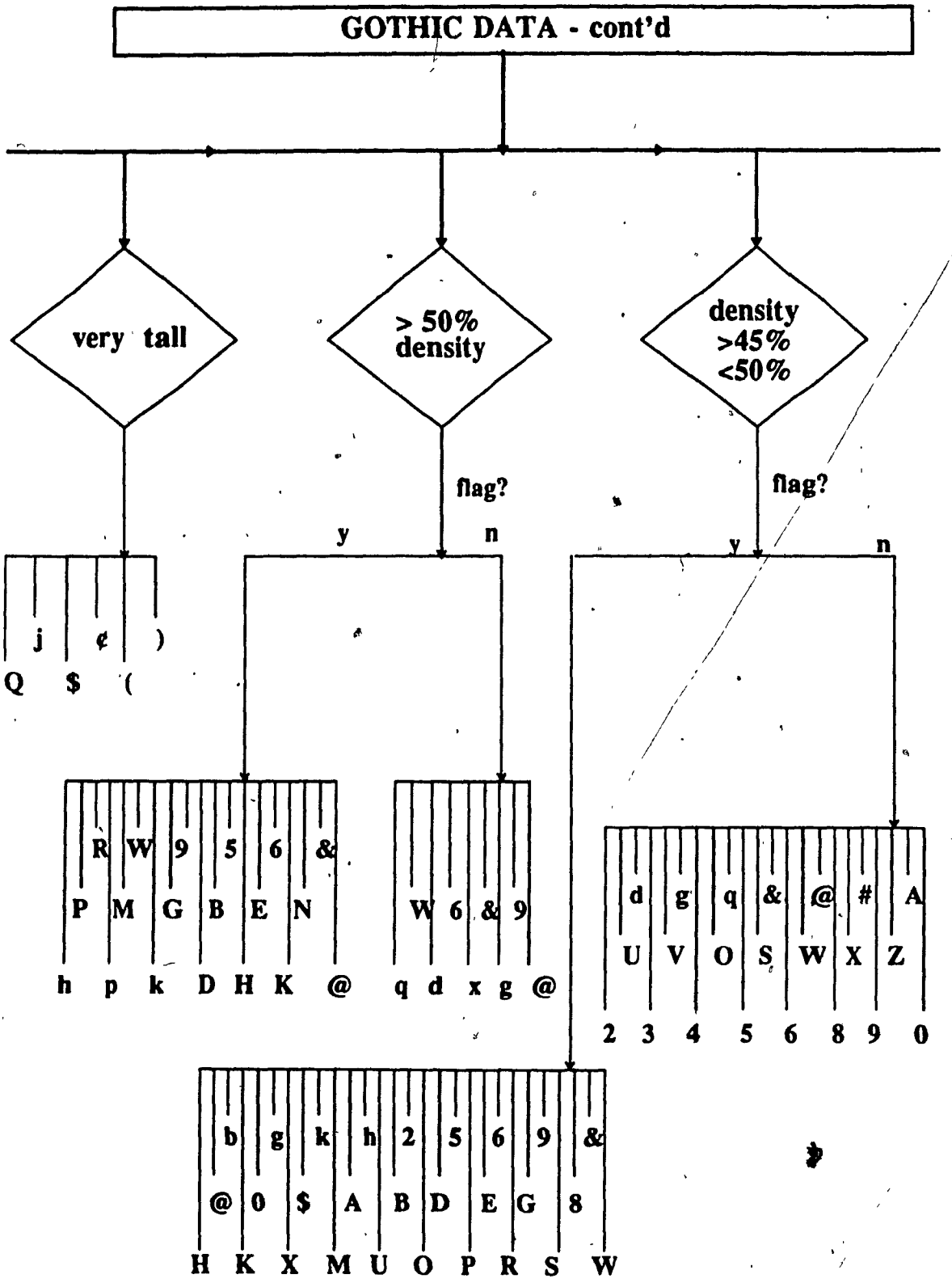
This secondary decision chart is also provided below.

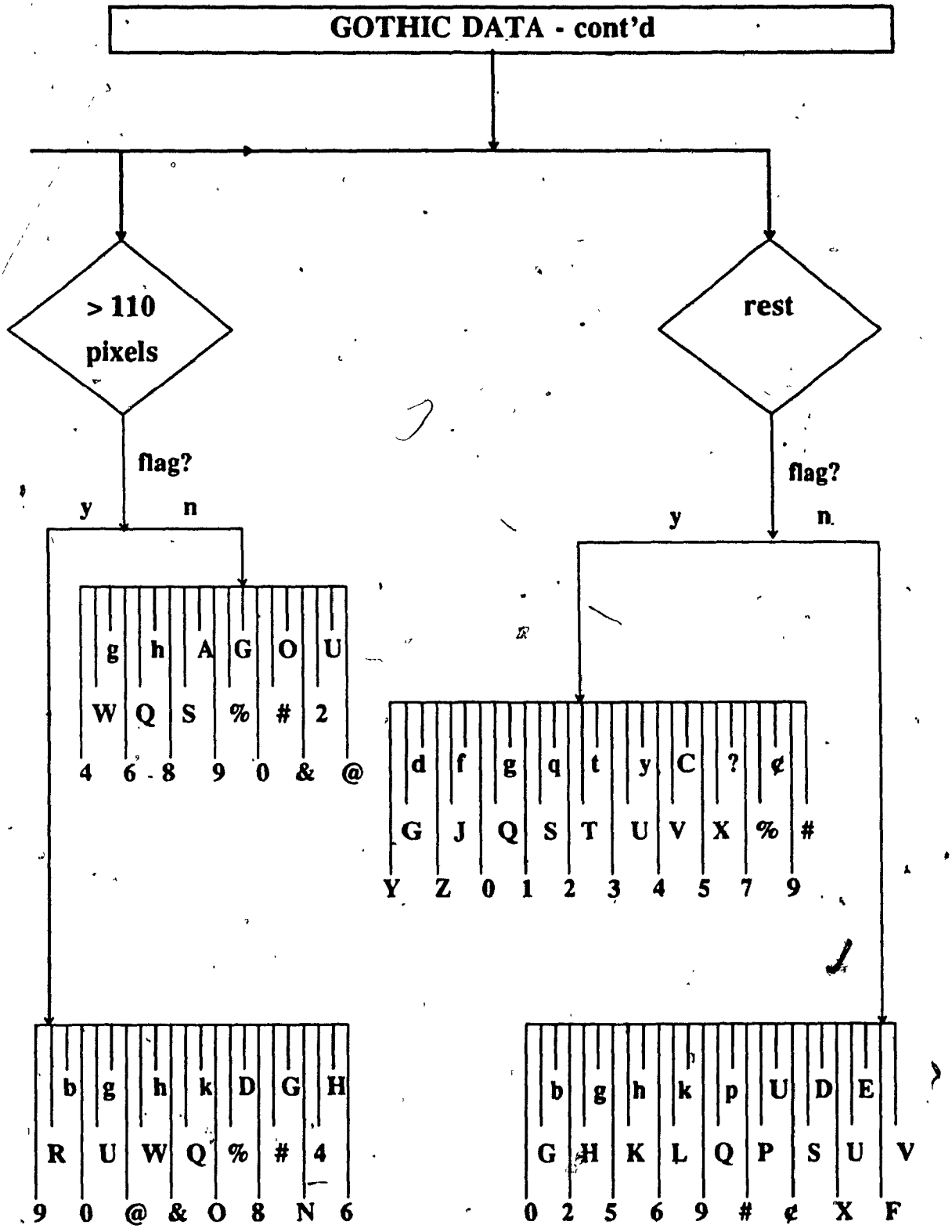
2.5.1 Gothic Font

2.5.1.1 Decision Tree




GOTHIC DATA - cont'd
narrow
lower case
< 35% density
flag?
y
n
< 50 pixels
y
n
z c : $
x v .
c v e r
s x u a
n z o
f y C 7 * %
j F P 9 ?
t L Y ¢ #
tall
j ( ) ;
k !) I ;
b / i 1
e l ? z
f y C 7 * %
j J / 9 ?
t T Y ¢ #


GOTHIC DATA - cont'd
very tall
> 50% density
density >45% <50%
flag?
flag?
y
n
y
n
j
¢
)
Q
$
(
R
W
9
5
6
&
P
M
G
B
E
N
h
p
k
D
H
K
@
W
6
&
9
q
d
x
g
@
d
g
q
&
@
#
A
U
V
O
S
W
X
Z
2
3
4
5
6
8
9
0
b
g
k
h
2
5
6
9
&
@
0
$
A
B
D
E
G
8
H
K
X
M
U
O
P
R
S
W

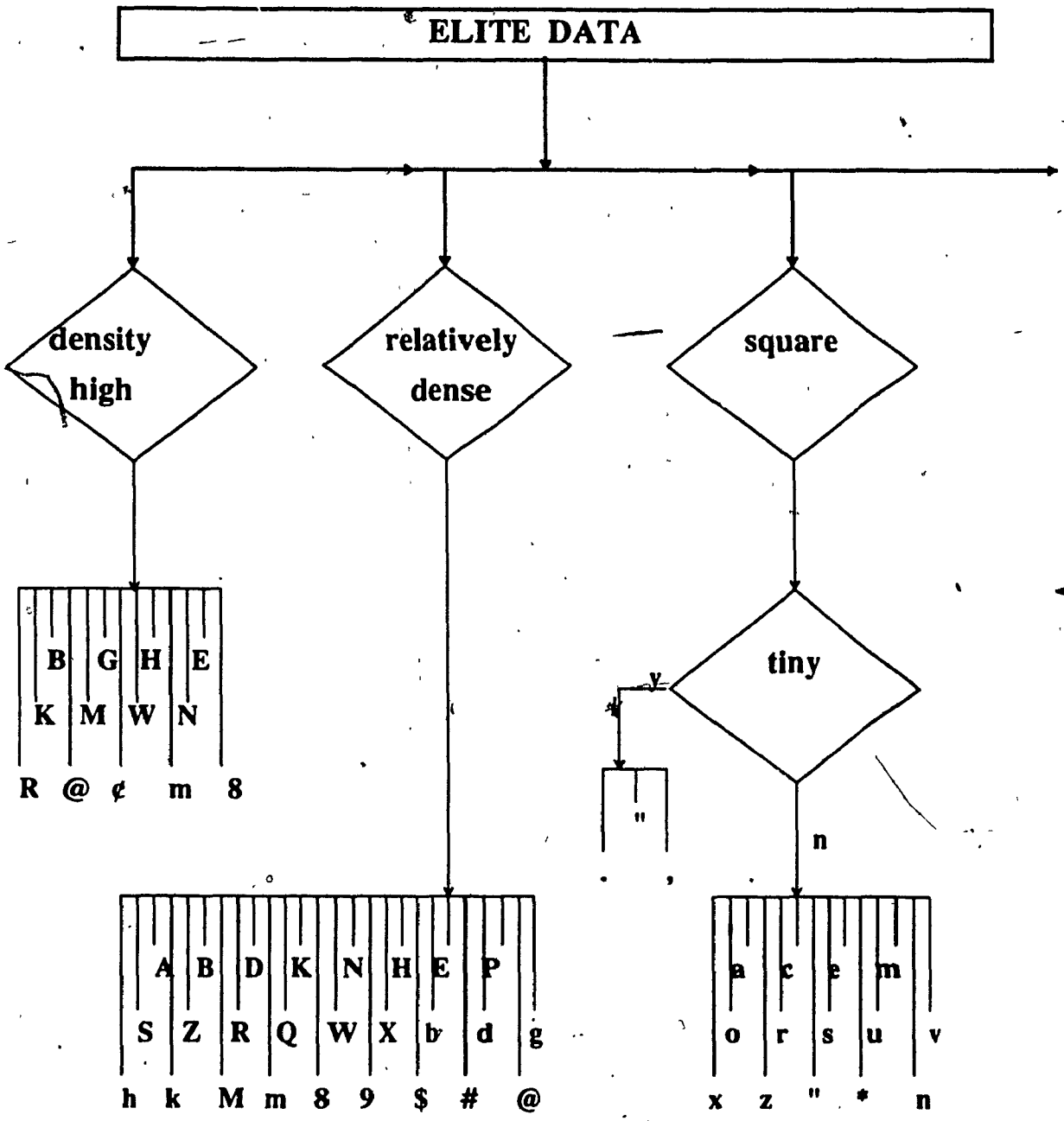

GOTHIC DATA - cont'd
> 110 pixels
rest
flag?
y
n
flag?
y
n
g h A G O U
W Q S % # 2
4 6 8 9 0 & @
d f g q t y C ? ¢
G J Q S T U V X % #
Y Z 0 1 2 3 4 5 7 9
b g h k D G H
R U W Q % # 4
9 0 @ & O 8 N 6
b g h k p U D E
G H K L Q P S U V
0 2 5 6 9 # ¢ X F

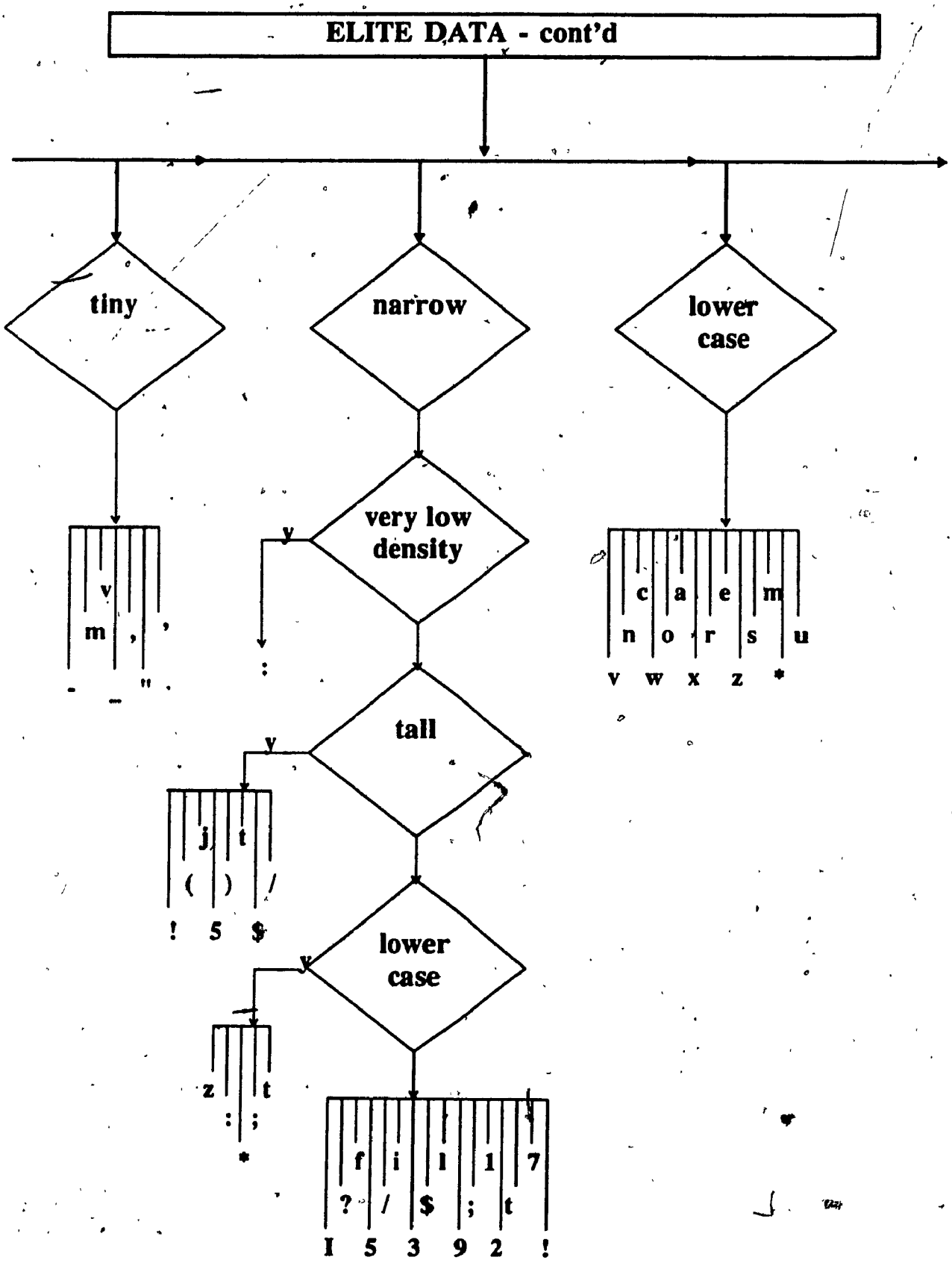
2.5.1.2 Secondary Decision Chart - Gothic

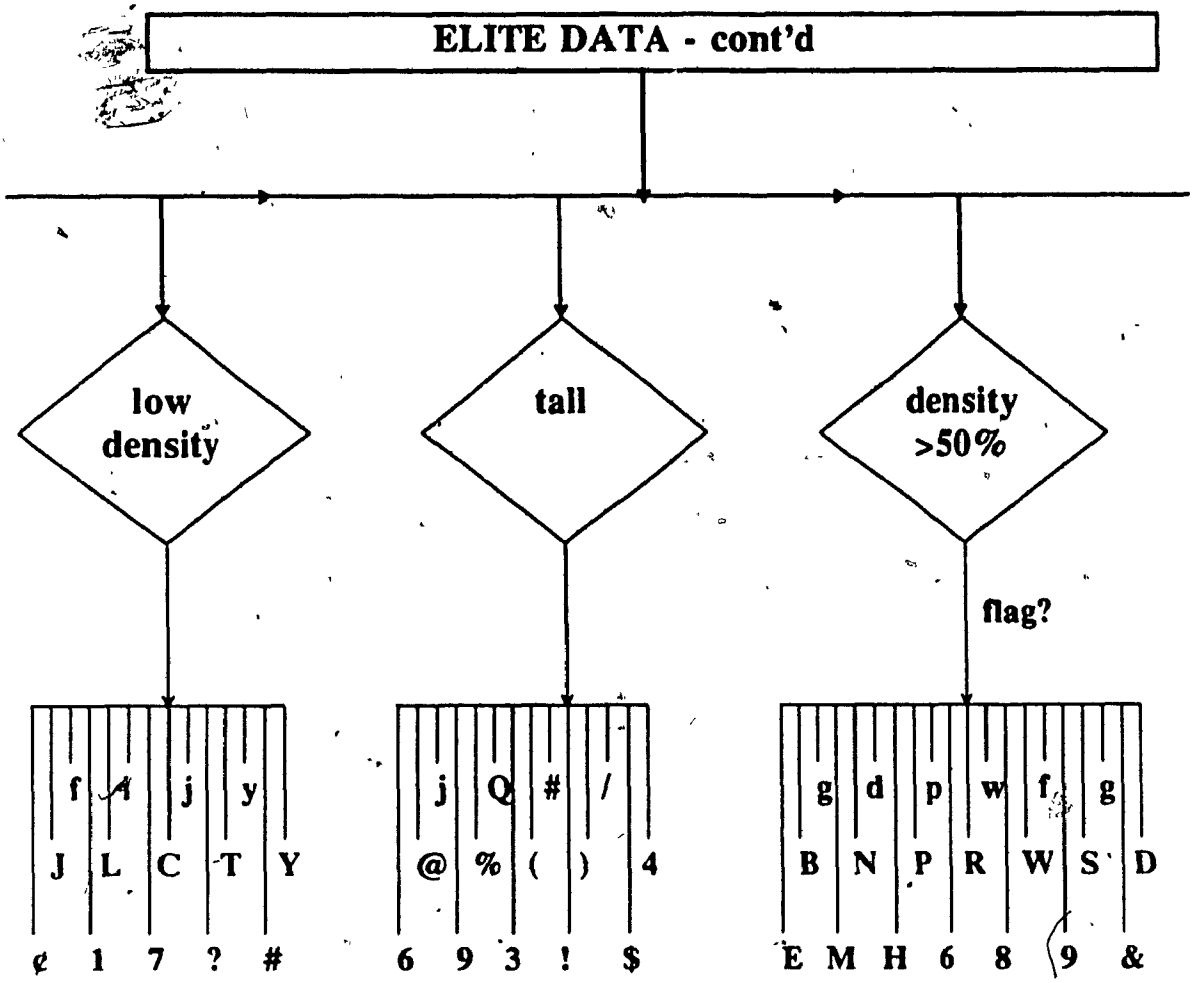
confused characters	criteria for distinguishing between them
M, W	density (in pixels) in rows approximately one third of height of character. Higher density implies 'M', otherwise result is 'W'
f, t	check for density (in pixels) of first three rows of character Higher density implies 'f', otherwise 't'.
u, n	density (in pixels) across top two rows of character. If density is high, result is 'n', otherwise it is 'u'.
9, g	check where character's top row is in relation to the data window. If character is high in window, set it as a '9', otherwise, 'g'.
B, 8	Check 3x3 matrix in top left hand corner of character. If it has fewer than 5 pixels, it is '8', otherwise, 'B'.
O, D	Check 3x3 matrix in top left hand corner of character. If it has fewer than 5 pixels, it is 'O', otherwise, 'D'.
0, D	Check 3x3 matrix in top left hand corner of character. If it has fewer than 5 pixels, it is '0', otherwise, 'D'.

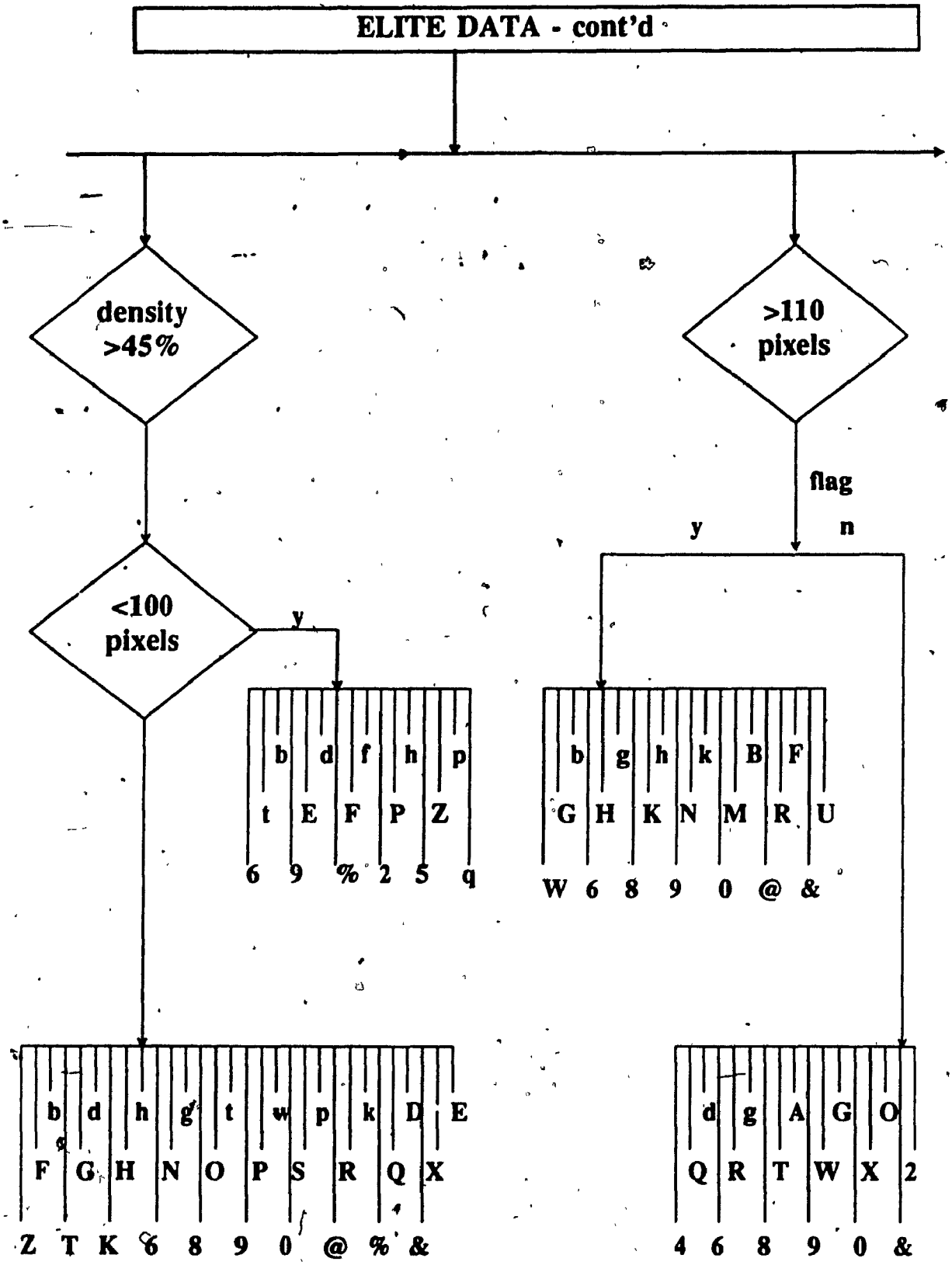
2.5.2 Elite Font

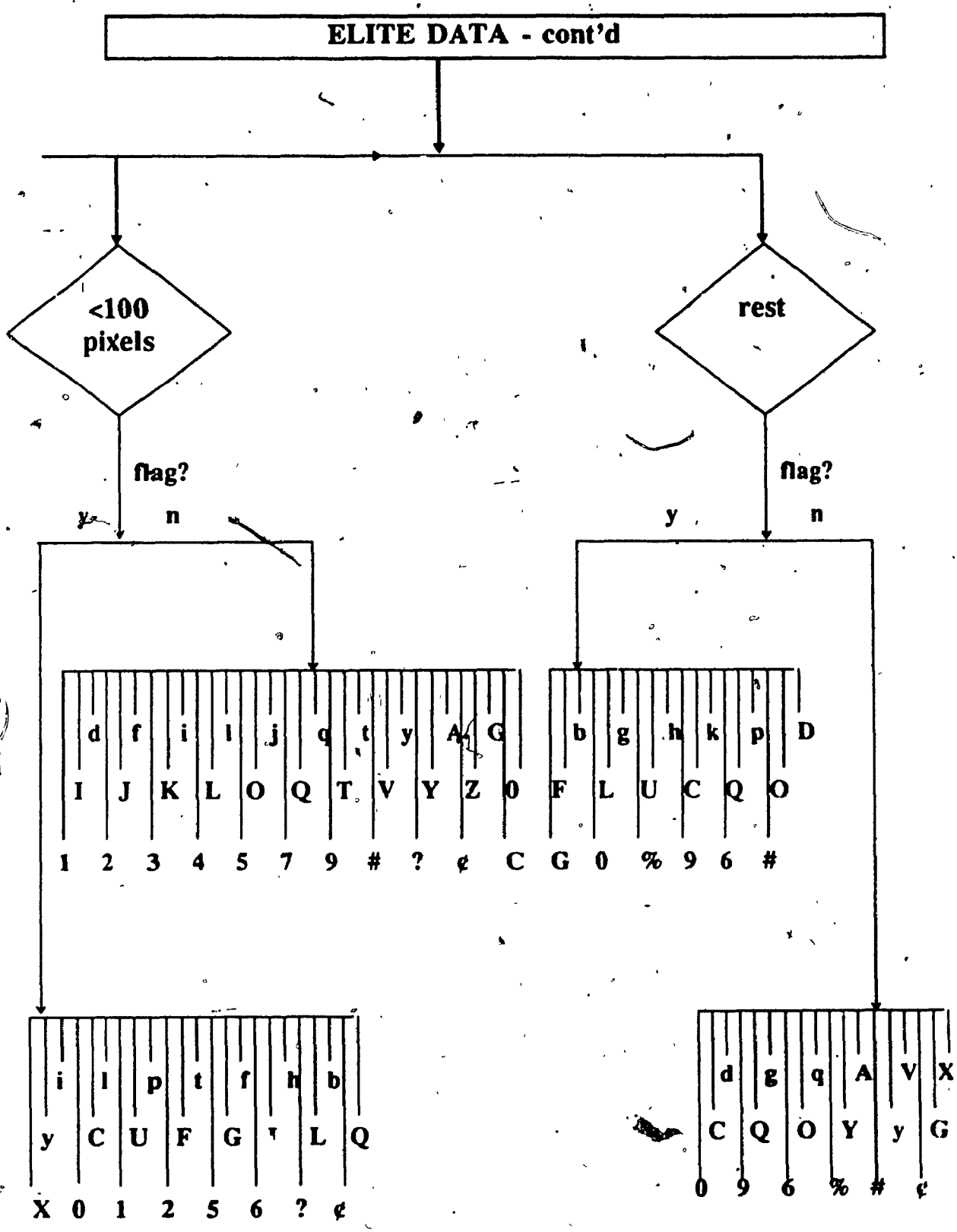
2.5.2.1 Decision Tree




ELITE DATA - cont'd
tiny
narrow
lower case
very low density
tall
lower case
v
m , '
- _ "
y
:
c a e m
n o r s u
v w x z *
y
j t
( ) /
! 5 $
z t
: ;
*
f i l 1 7
? / $; t
I 5 3 9 2 !


ELITE DATA - cont'd
low density
tall
density >50%
flag?
f A j y
J L C T Y
¢ 1 7 ? #
j Q # /
@ % ( ) 4
6 9 3 ! $
g d p w f g
B N P R W S D
E M H 6 8 9 &


ELITE DATA - cont'd
density >45%
>110 pixels
flag
y
n
<100 pixels
y
b d f h p
t E F P Z
6 9 % 2 5 q
b g h k B F
G H K N M R U
W 6 8 9 0 @ &
b d h g t w p k D E
F G H N O P S R Q X
Z T K 6 8 9 0 @ % &
d g A G O
Q R T W X 2
4 6 8 9 0 &

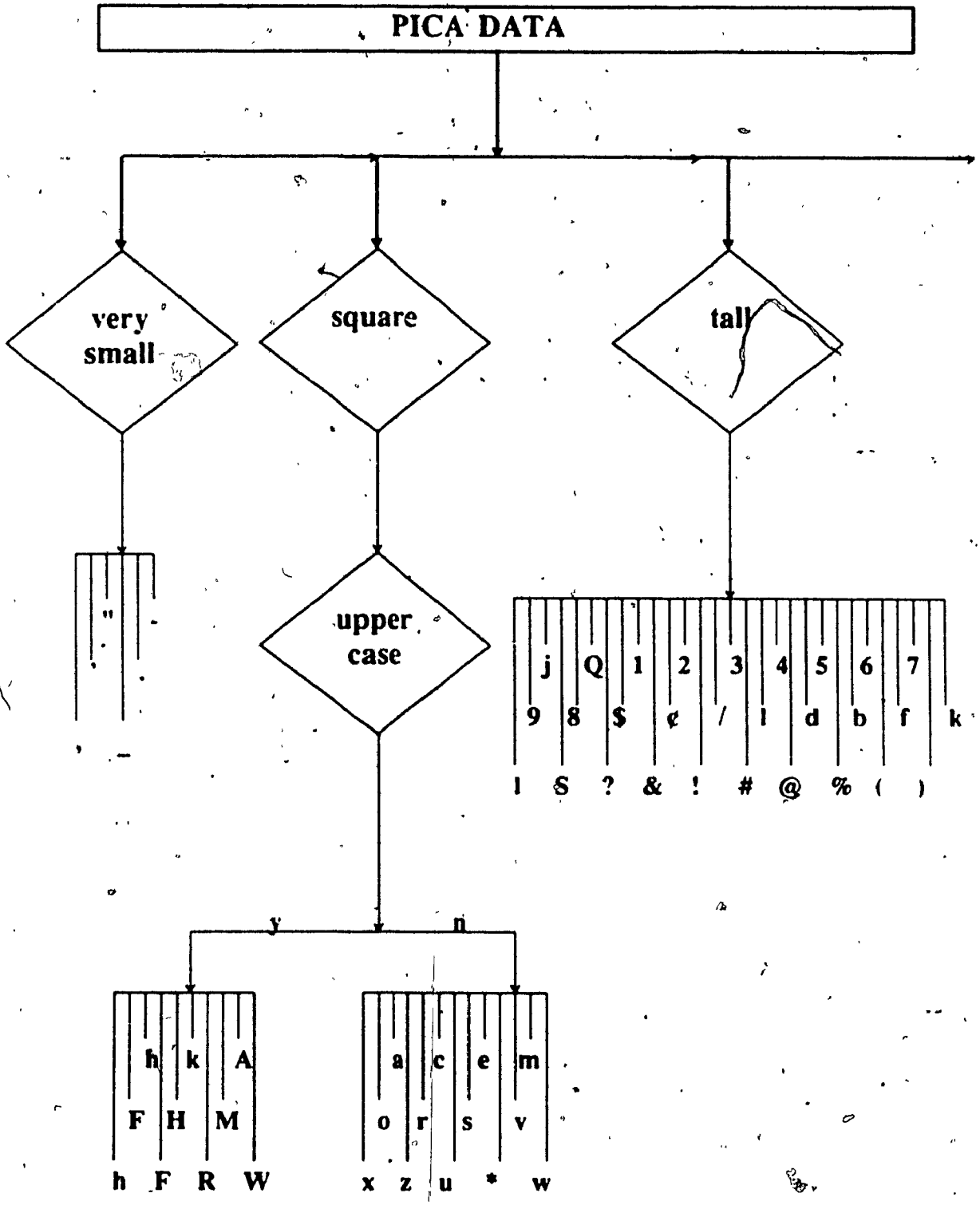

ELITE DATA - cont'd
<100 pixels
rest
flag?
y
n
flag?
y
n
d f i l j q t y A G
I J K L O Q T V Y Z 0
1 2 3 4 5 7 9 # ? ¢ C
b g h k p D
F L U C Q O
G 0 % 9 6 #
i l p t f h b
y C U F G L Q
X 0 1 2 5 6 ? ¢
d g q A V X
C Q O Y y G
0 9 6 % # ¢

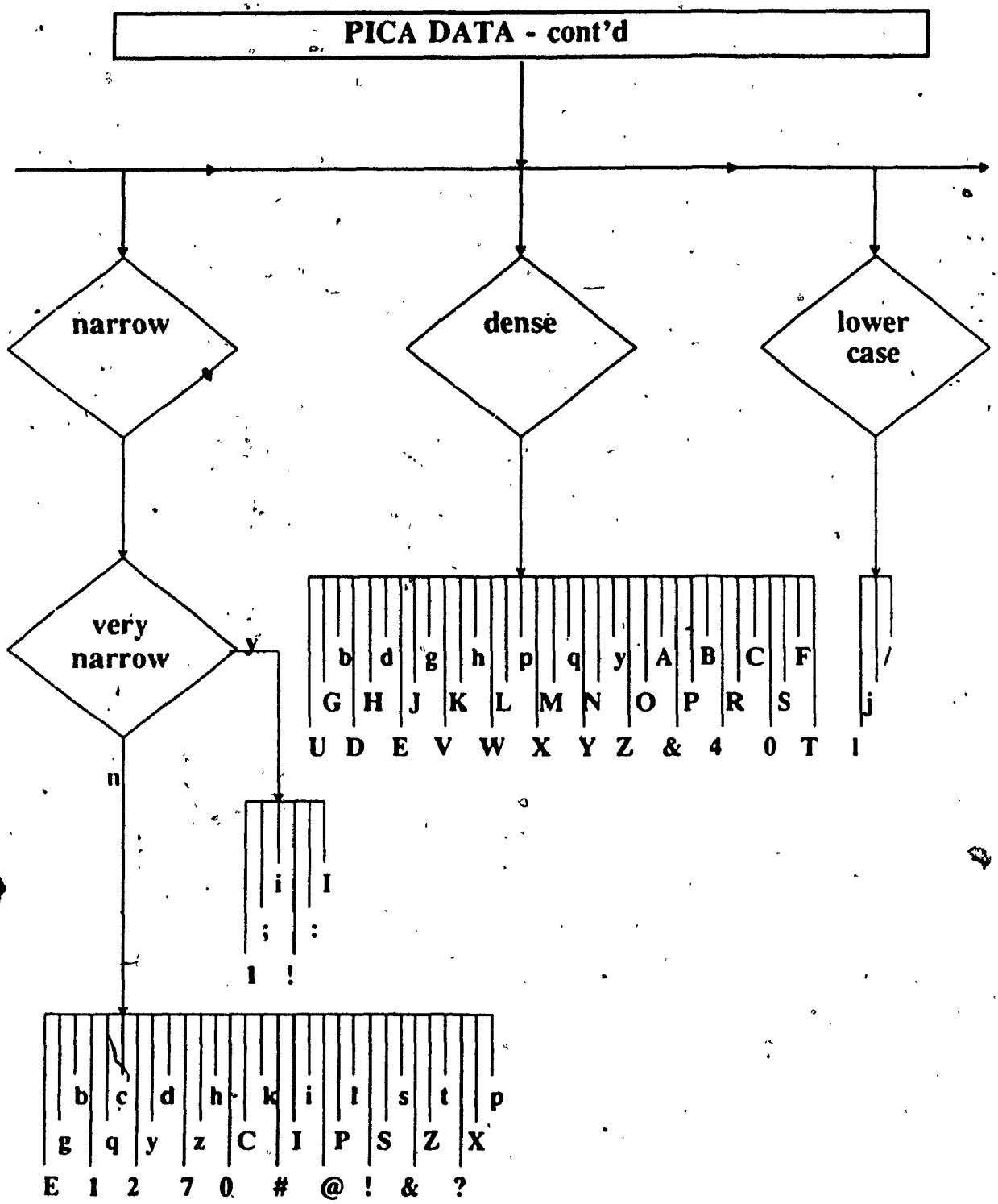
2.5.2.2 Secondary Decision Chart - Elite

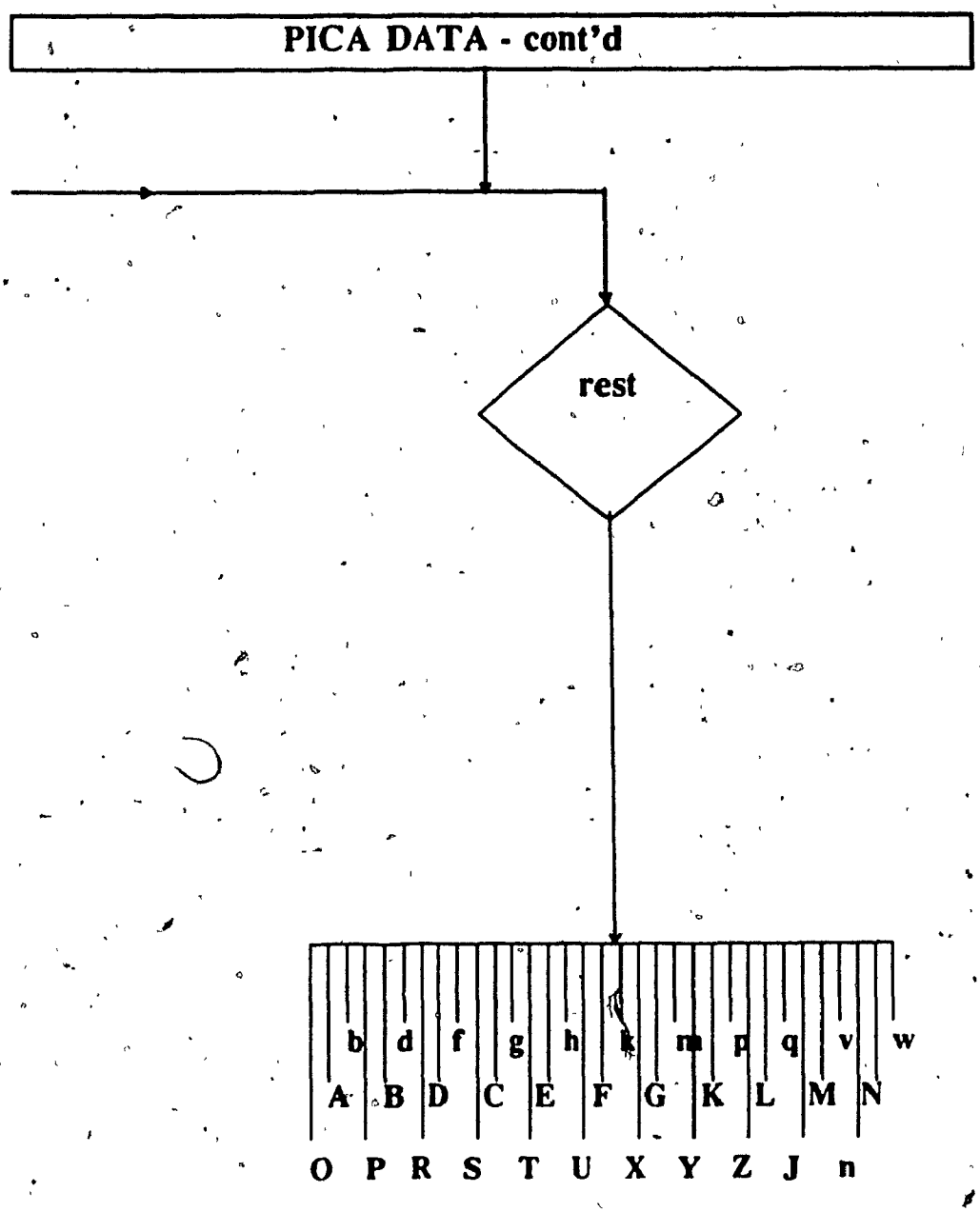
confused characters	criteria for distinguishing between them
M W	density (in pixels) in rows approximately one third of height of character. Higher density implies 'M', otherwise result is 'W'
p q	check for density (in pixels) of first three columns of character Higher density implies 'p', otherwise 'q'.
b d	check for density (in pixels) of first three columns of character Higher density implies 'b', otherwise 'd'.
a s	check first 3 columns midway down the character. If it is dense there, result is 's', otherwise it is 'a'
(j	check density of last two rows. If it is dense then the character is a 'j', otherwise it is '('.
U V	check width of character at bottom rows. If is quite a bit narrower than the width of the character, it's a 'V', else 'U'.
i l	look for serif under dot of the 'i'. If it is present, the character is an 'i', otherwise it is an 'l'
B E	look for density of pixels where 'B' closes in the center on the right of the character. If dense it's a 'B', otherwise 'E'
M N	look for density of pixels in center of character. If very dense, character is an 'M', otherwise it's an 'N'.
u v	check width of character at bottom rows. If is quite a bit narrower than the width of the character, it's a 'v', else 'u'.
', `	if character position is high in data window, then it is the ''' otherwise it is the ','
T Y	check crossing numbers (horizontal) for first 3 rows of the character. A high number implies a 'Y', a low one, a 'T'.
P F	check crossing number at serifs of second horizontal line in 'F' if crossing number is high, it is an 'F', else it is a 'P'
O D	check 3x3 matrix at top left hand corner of character. If it is dense, character is 'D', otherwise 'O'
U D	check crossing numbers (horizontal) for first 3 rows of the character. A high number implies a 'U', a low one, a 'D'.

2.5.3 Pica Font

2.5.3.1 Decision Tree




PICA DATA - cont'd
narrow
dense
lower case
very narrow
y
n
b d g h p q y A B C F /
G H J K L M N O P R S j
U D E V W X Y Z & 4 0 T I
i I
; :
1 !
b c d h k i l s t p
g q y z C I P S Z X
E 1 2 7 0 # @ ! & ?

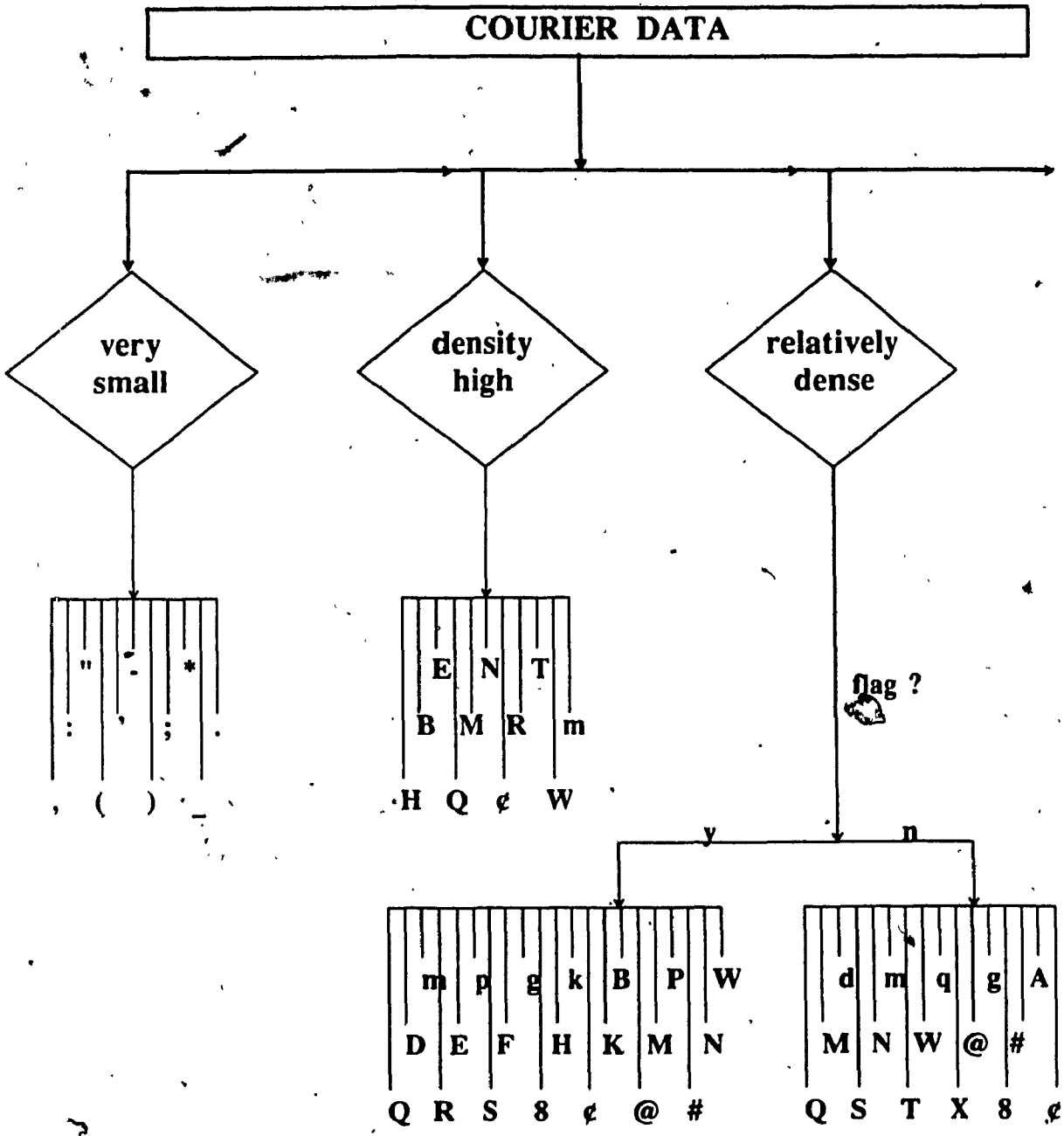

PICA DATA - cont'd
rest
b d f g h k m p q v w
A B D C E F G K L M N
O P R S T U X Y Z J n

2.5.3.2 Secondary Decision Chart - Pica

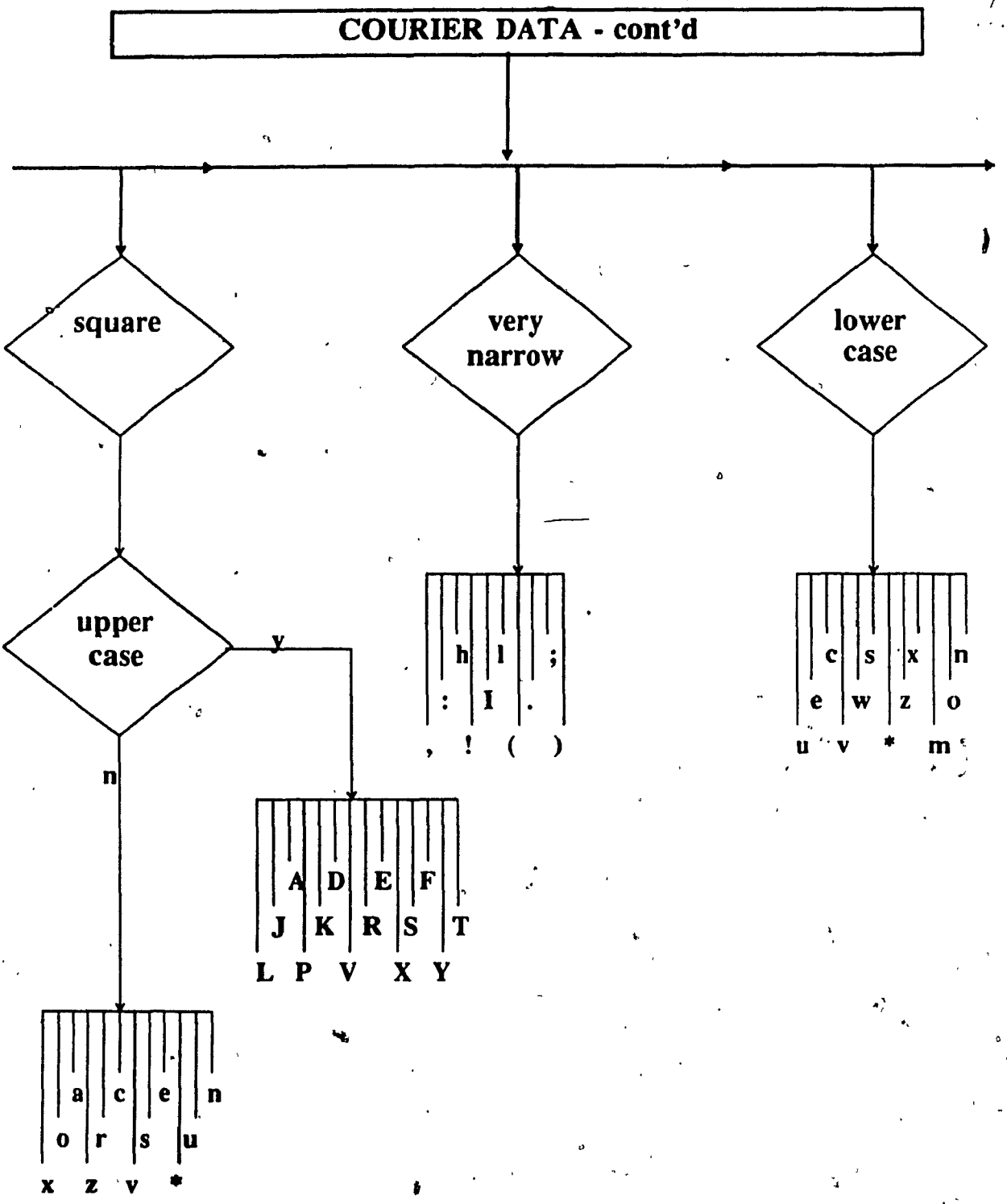
confused characters	criteria for distinguishing between them
b, d	density (in pixels) first three columns of character. If density is high, result is 'b', otherwise it is 'd'.
p, q	density (in pixels) first three columns of character. If density is high, result is 'p', otherwise it is 'q'.
B, g	check where character's top row is in relation to the data window. If character is high in window, set it as a 'B', otherwise, 'g'.
O, D	Check 3x3 matrix in top left hand corner of character. If it has fewer than 5 pixels, it is 'O', otherwise, 'D'.

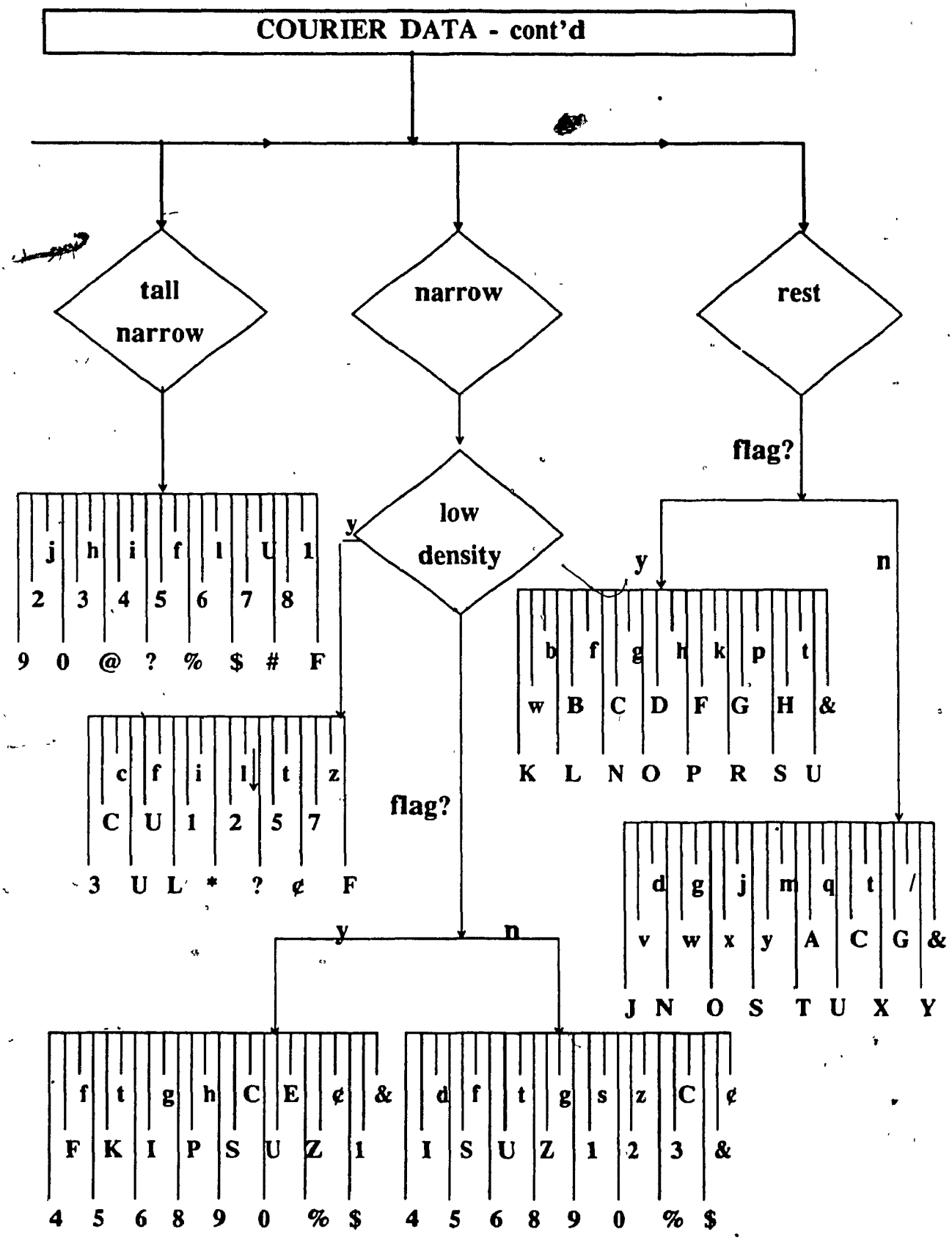
2.5.4 Courier Font

2.5.4.1 Decision Tree



COURIER DATA - cont'd




COURIER DATA - cont'd
tall narrow
narrow
rest
flag?
low density
y
n
y
j h i f l U 1
2 3 4 5 6 7 8
9 0 @ ? % $ # F
b f g h k p t
w B C D F G H &
K L N O P R S U
c f i l t z
C U 1 2 5 7
3 U L * ? ¢ F
flag?
y
n
d g j m q t /
v w x y A C G &
J N O S T U X Y
f t g h C E ¢ &
F K I P S U Z 1
4 5 6 8 9 0 % $
d f t g s z C ¢
I S U Z 1 2 3 &
4 5 6 8 9 0 % $

2.5.4.2 Secondary Decision Chart - Courier

confused characters	criteria for distinguishing between them
p, q	density (in pixels) first three columns of character. If density is high, result is 'p', otherwise it is 'q'.
L, J	density (in pixels) first three columns of character. If density is high, result is 'L', otherwise it is 'J'.
5, $	Check 3x3 matrix in top left hand corner of character. If it has fewer than 5 pixels, it is '$', otherwise, '5'.
O, D	Check 3x3 matrix in top left hand corner of character. If it has fewer than 5 pixels, it is 'O', otherwise, 'D'.

3.0 Results

The results for each font are presented below.

For each font tabular results are first presented. These results include the number of characters tested and the number of complete data sets used. A data set is defined as a complete set of the characters recognized by the system. The four fonts tested were Gothic (12 pitch), Elite (12 pitch), Pica (10 pitch) and Courier (10 pitch).

The tables provide the success rates of the different fonts together with the average speed to recognize the characters.

Note: all timing results were produced on a Philips PC. These speeds would differ on a true IBM PC.

Recognition results varied between 98.1% (positive recognition) for the Gothic font, to 98.8% for that of the Pica font. The calculation of the recognition rates was performed in the following manner:

$$\frac{\text{\# positive recognitions + \# reasonable recognitions}}{\text{total number of characters tested}} = \text{\% success}$$

Reasonable recognitions include those confusions that cannot be overcome due to factors beyond the scope of this project. Such confusions include the confusion between 'l' (lower case L) and '1' (one) in the Pica and Courier fonts. These characters are identical in all aspects, even to the human eye. Recognition of these must be done within the context of text, which cannot be reproduced here. Another such case is the confusion of 'O' (upper case 'o') and '0' (zero) in Gothic and Elite.

Other 'misrecognitions' that are included in this category are those that are the result of imperfect data. Appendix 'D' includes a typical example of such a case. The character 'b' was cut off in some instances (either due to faulty scanning procedure or a slightly damaged printwheel producing the data). As a result of this, the cut-off 'b's were often recognized as 'h'. This cannot be considered as an error. A similar situation arises when 'Q' is mistaken for 'O' when the tail of the 'Q' is not present.

The second set of results included, provide the elapsed time to produce recognition of the characters. The timing results were produced in the following manner. The time to process an entire data set (83 characters) was recorded. This time was divided by 83 to provide an average time per

character. (All timing results are in seconds). These results included the time to load the program, read in the data and perform any required preprocessing. In order to provide more meaningful results, recordings were made of the time to perform each of these three categories:

i) to load the program and read in the data;

ii) to preprocess the data by means of thinning and filling techniques;

iii) to actually perform the decision making.

The time to read the data was by far the most time consuming of these. This time however, is not important here because it has nothing to do with the actual recognition system. In any such system the data would be accessible directly to the recognition system from the scanner without the very costly (in terms of time) intermediate requirement of having to read in approximately 1600 bytes of information from the disk for each character.

In addition to the tables provided, there are also confusion tables for each font. These tables provide a summary of all the misrecognitions that were produced for each font. The vertical axis represents the actual identity of the characters, and the horizontal axis represents the error produced. Lines have been drawn in to facilitate the reading of the graph. In order to further simplify these tables, correct recognitions are not included. The number of correct recognitions per character may be assumed by the number of data sets tested for each font.

It should be noted that these so-called 'reasonable recognitions' are included in the confusion tables.

3.1 Gothic Font Results

Recognition Results

#chars tested	# data sets	# correct	# of errors	# rejected	% success	average time (sec/ dataset)*	average time (sec/ char)*
1660	20	1626	34**	0	98.0	121.7 +/- 0.5	1.47 +/- .01

* *includes time it takes to read in data, scale it and provide results.*

** *errors calculated taking into consideration two factors: the data used contained poor digitizations of the character 'b'. In this case the characters were cut off at the bottom (see Appendix 'D') Thus misrecognition of 'b' as 'h' was not considered an error.*

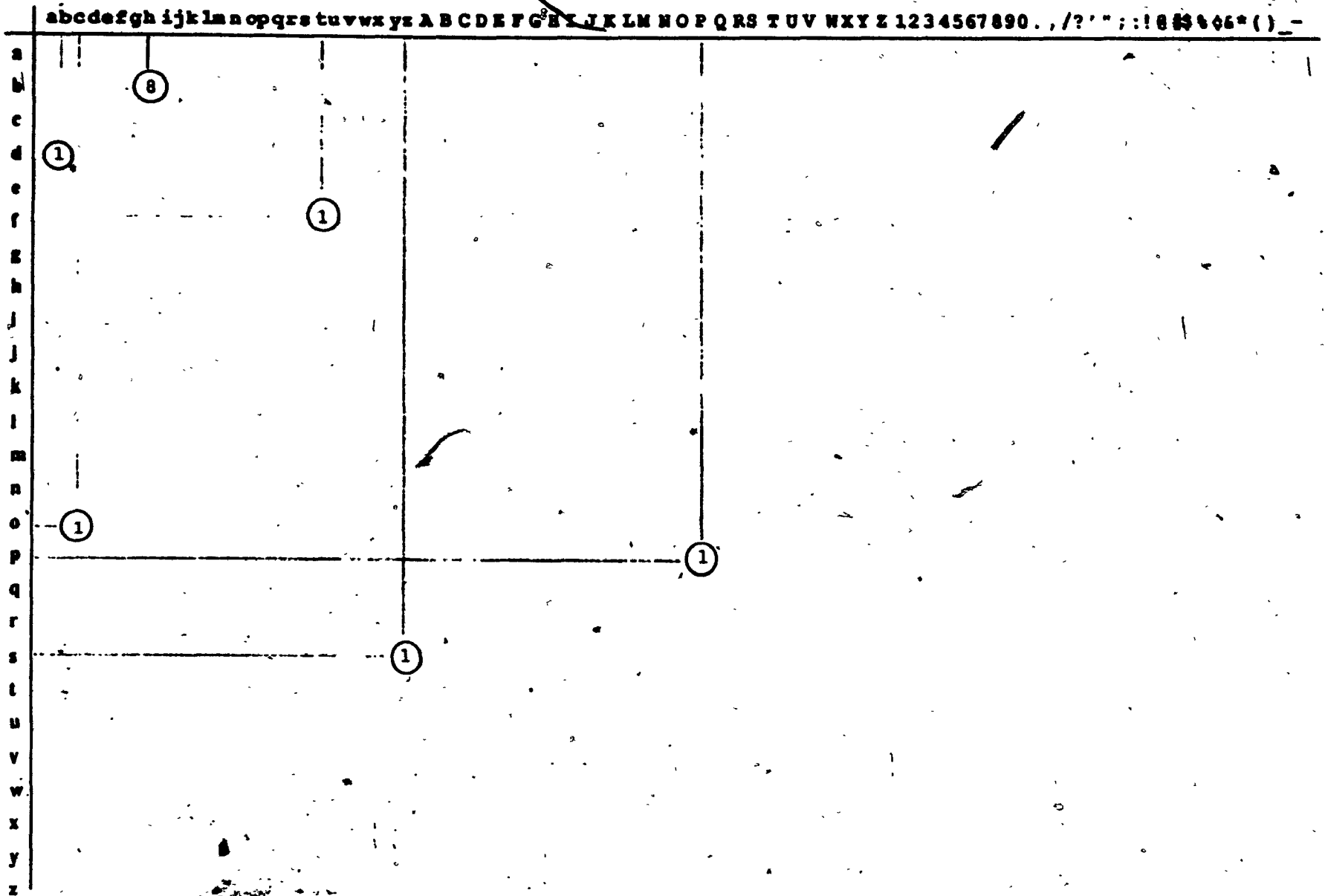
The second consideration was the confusion between 'O' and '0'. In the Gothic font there is no distinguishable difference between the two and thus these cases cannot be considered erroneous.

Timing Results (seconds per character)

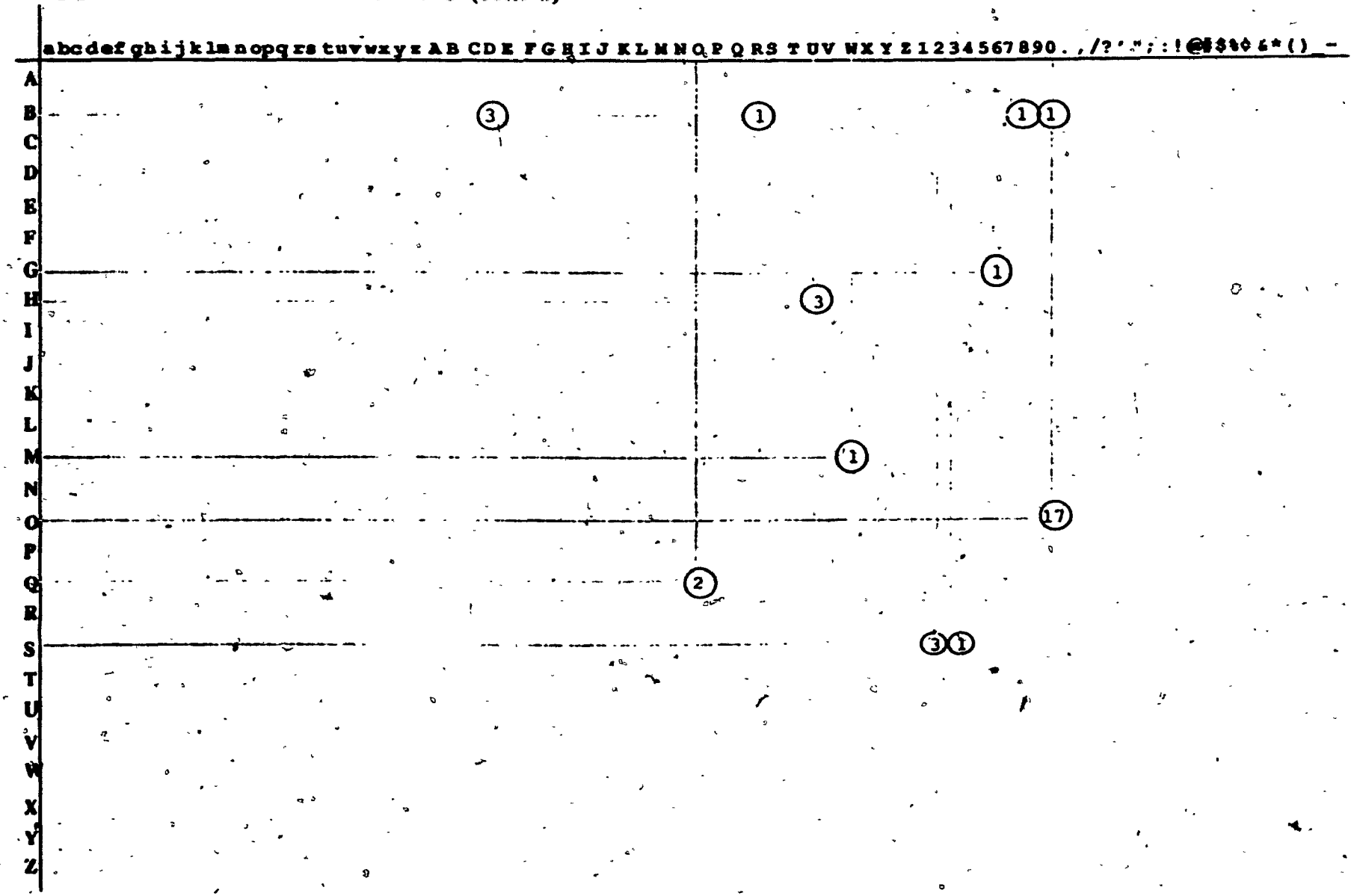
total time *	system overhead (reading data,	thinning, filling	decision making
1.47 +/- .01	1.01 +/- .0012	0.35 +/- 0.25	0.13 +/- 0.013

* *includes time it takes to read in data, scale it and provide results.*

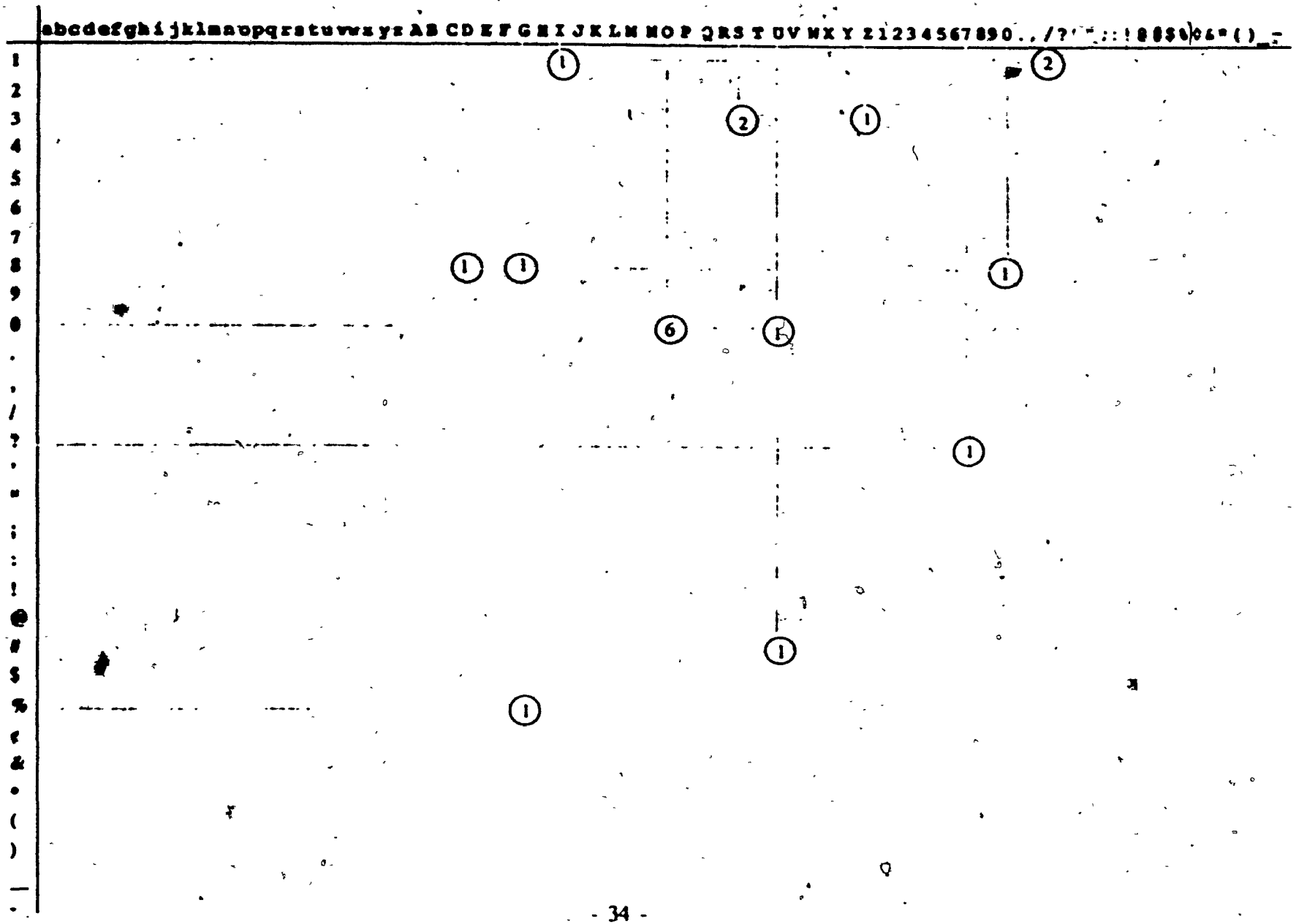
3.1.1 CONFUSION TABLE - GOTHIC



CONFUSION TABLE - GOTHIC (cont'd)



CONFUSION TABLE - GOTHIC (cont'd)



3.2 Elite Font Results

Recognition Results

#chars tested	# data sets	# correct	# of errors	# rejected	% success	average time (sec/ dataset)*	average time (sec/ char)*
1660	20	1638	22**	0	98.7**	120.1 +/- 0.5	1.4418 +/- .01

* *includes time it takes to read in data, scale it and provide results.*

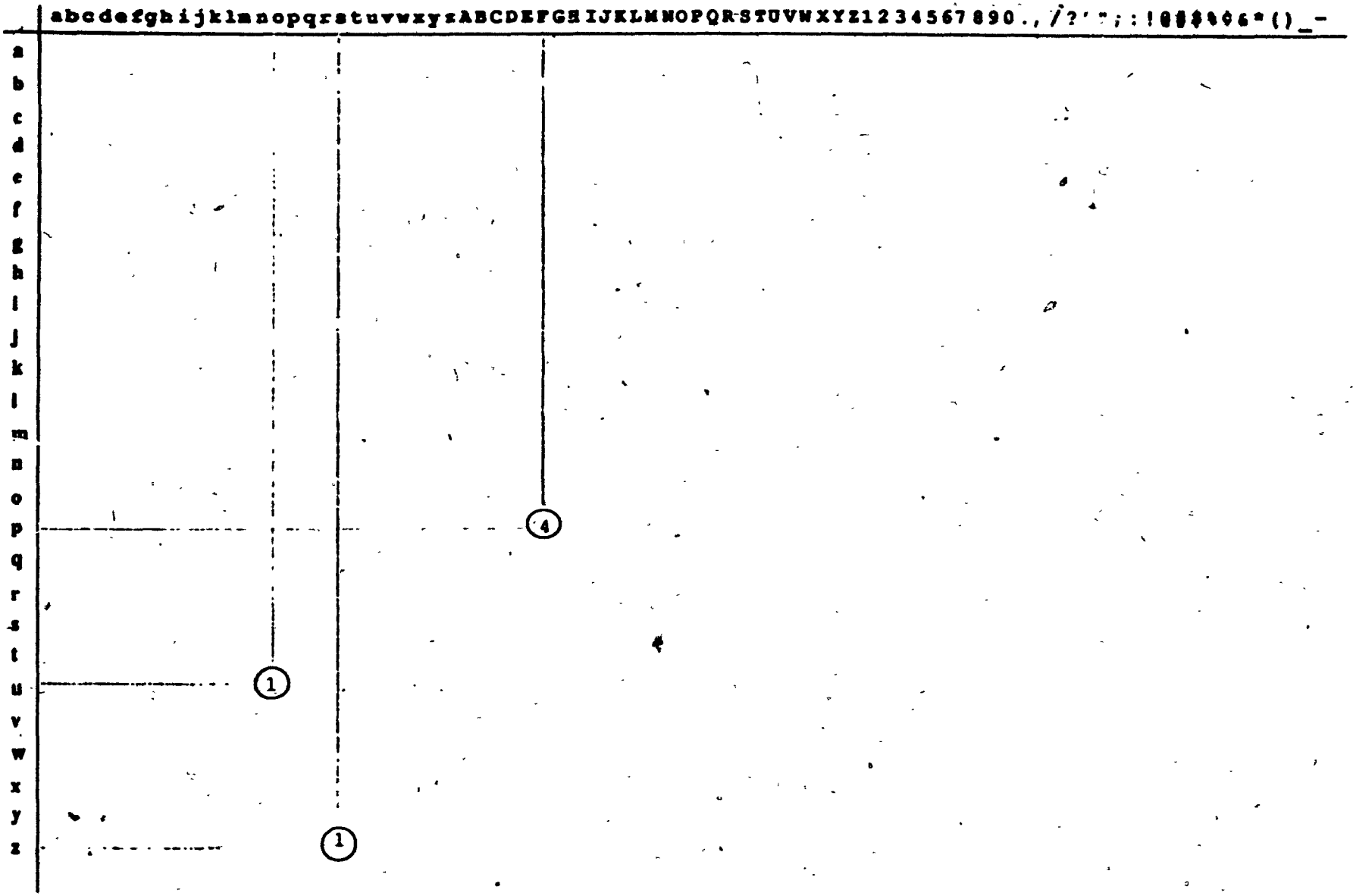
** *errors calculated with the assumption that the characters 'O' and '0' are indistinguishable as such in the Elite font. (10 such cases arise in the testing of the data). Another problem arises with the letter 'Q'. The data provided does not include a truly recognizable 'Q' - all the data is cut off at the bottom (see Appendix 'C' for examples of the data). For this reason, the recognition of a 'Q' as either an 'O' or '0' was not considered an error. If they were included in the error total, the % success would drop to 96.9%.*

Timing Results (seconds per character)

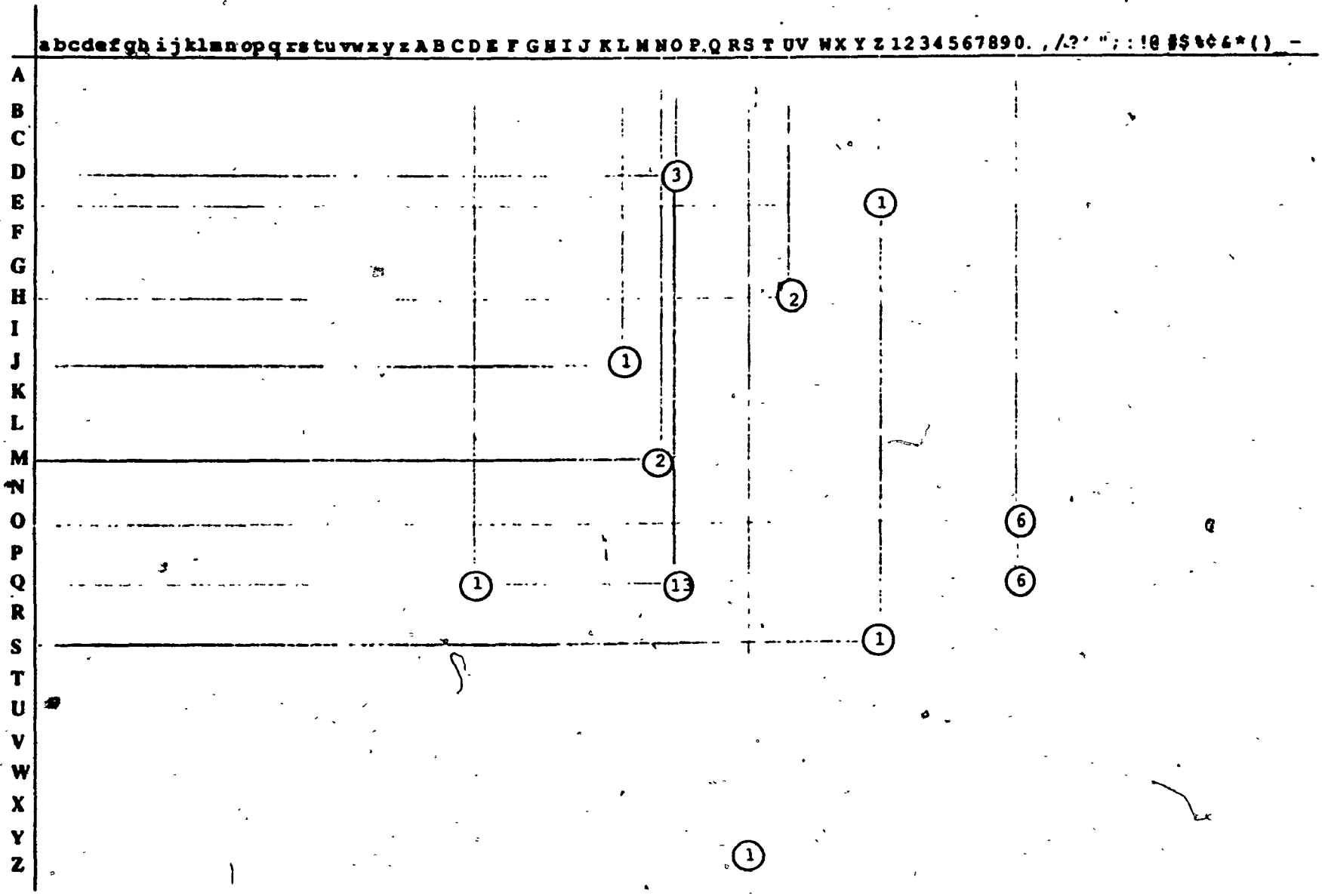
total time*	system overhead (reading data,	thinning, filling	decision making
1.44 +/- .01	1.01 +/- .0012	0.35 +/- 0.25	0.10 +/- 0.013

* *includes time it takes to read in data, scale it and provide results*

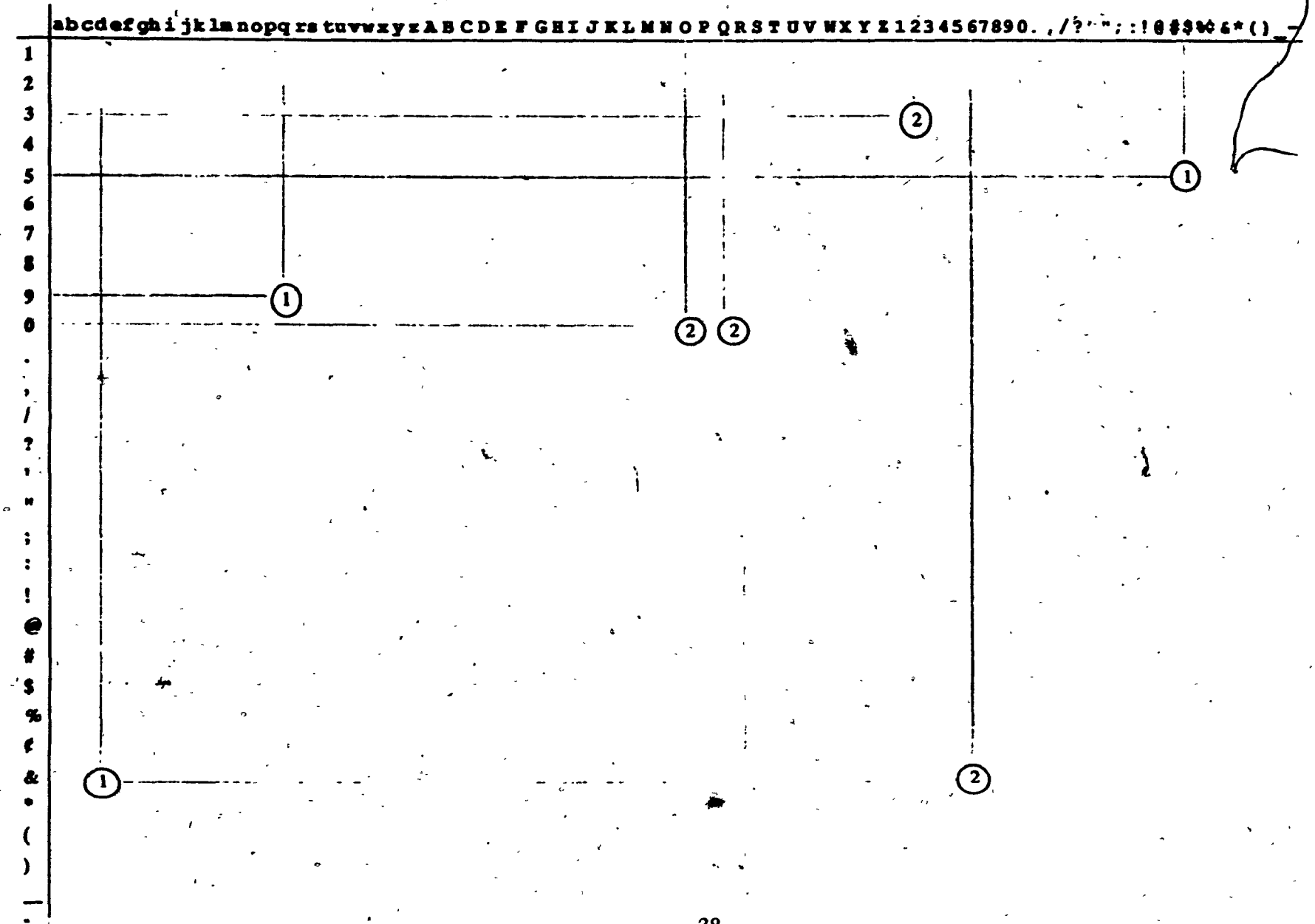
3.2.1 CONFUSION TABLE - ELITE



CONFUSION TABLE - ELITE (cont'd)



CONFUSION TABLE - ELITE (cont'd)



3.3 Pica Font Results

Recognition Results

#chars tested	# data sets	# correct	# of errors	# rejected	% success	average time (sec/ dataset)*	average time (sec/ char)*
1577	19	1558	19**	0	98.8**	131.0 +/- 2.9	1.58 +/- .03

* *includes time it takes to read in data, scale it and provide results.*

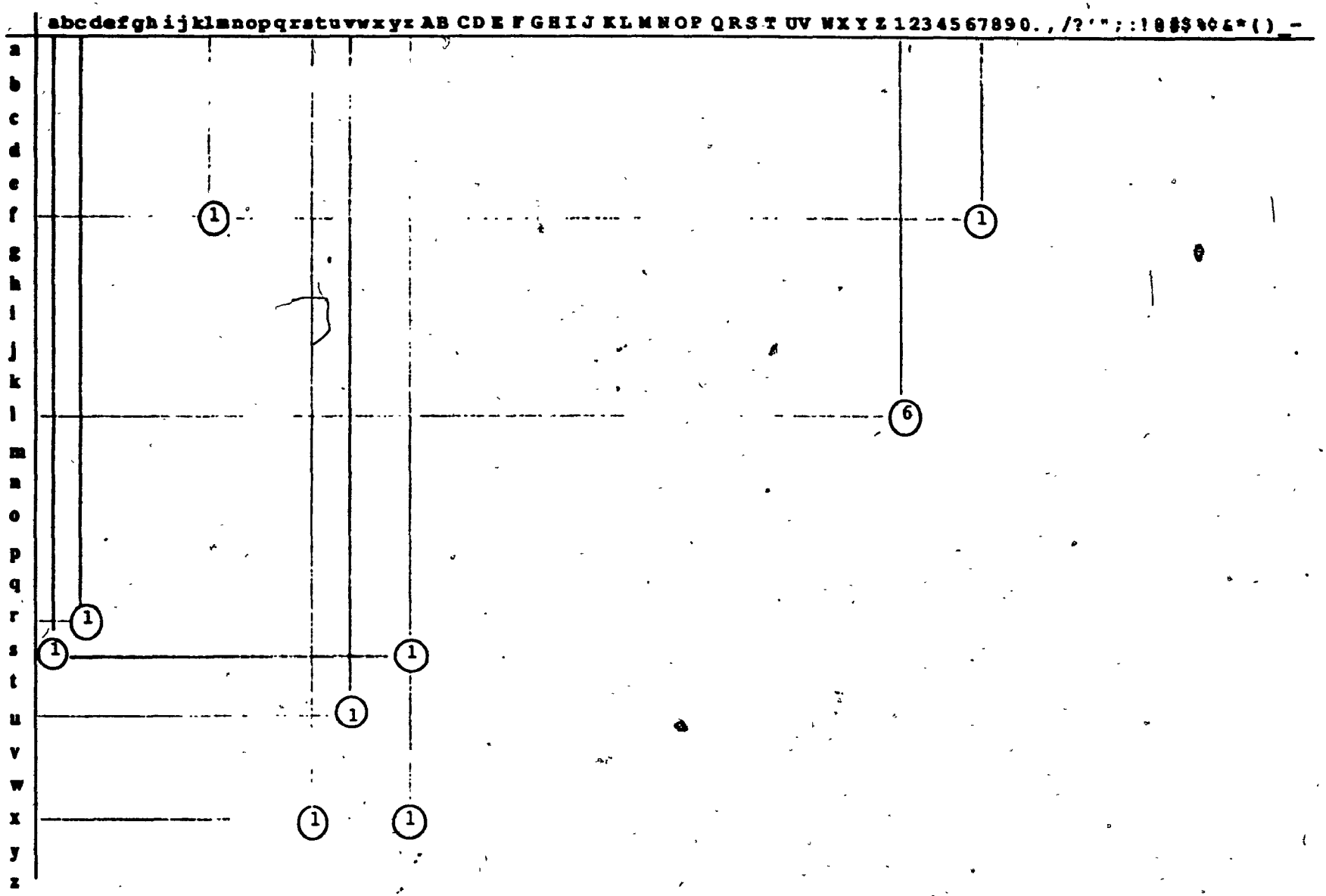
** *errors calculated with the assumption that the characters '1' and 'l' are indistinguishable in the Pica font. If misrecognitions between these two characters are included the number of errors reaches 29 and the percent success becomes 98.2%*

Timing Results (seconds per character)

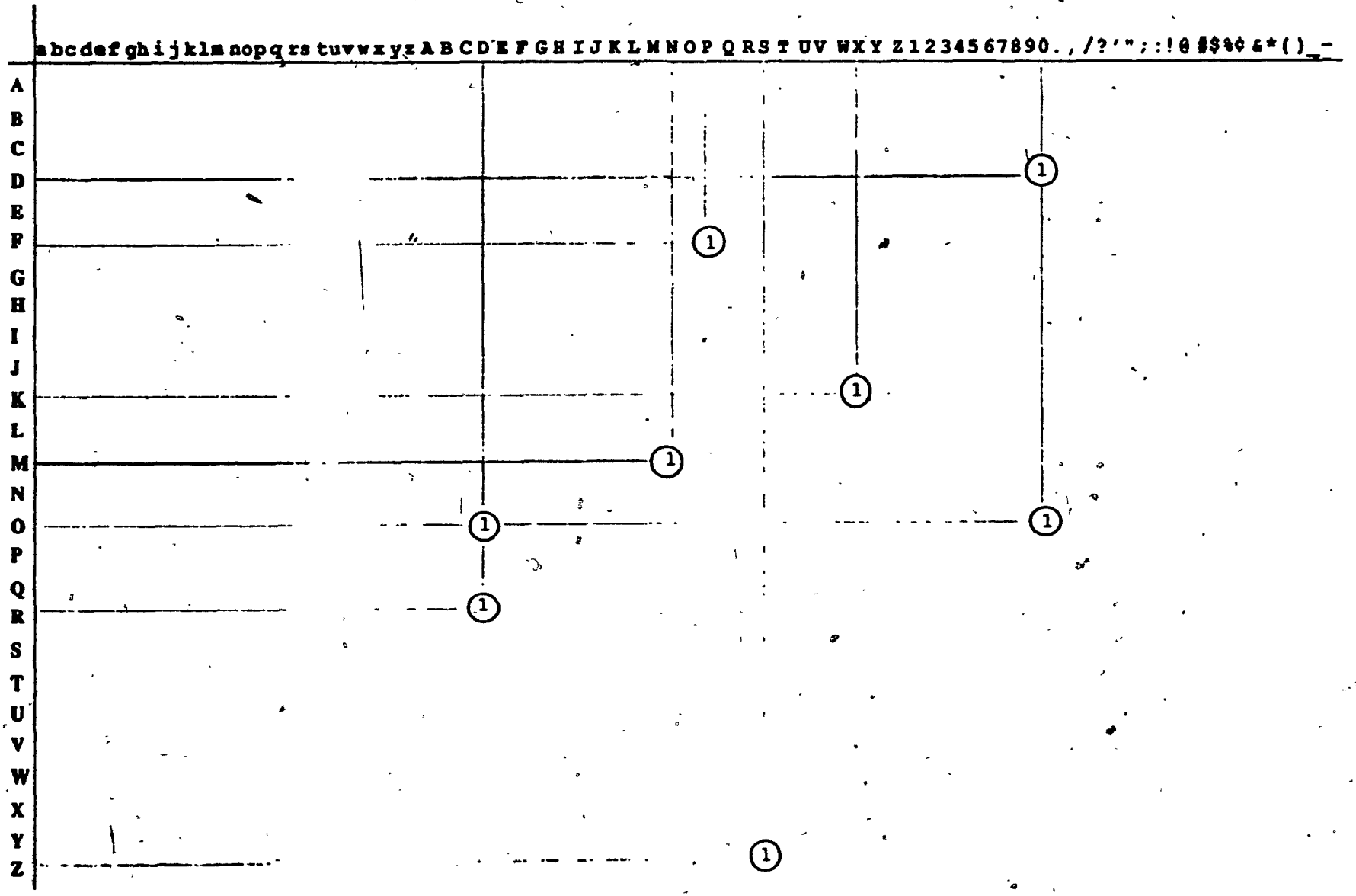
total time* (sec/char)	system overhead (reading data,	thinning, filling	decision making
1.58 +/- .03	1.01 +/- .0012	0.35 +/- 0.25	0.22 +/- 0.018

* *includes time it takes to read in data, scale it and provide results.*

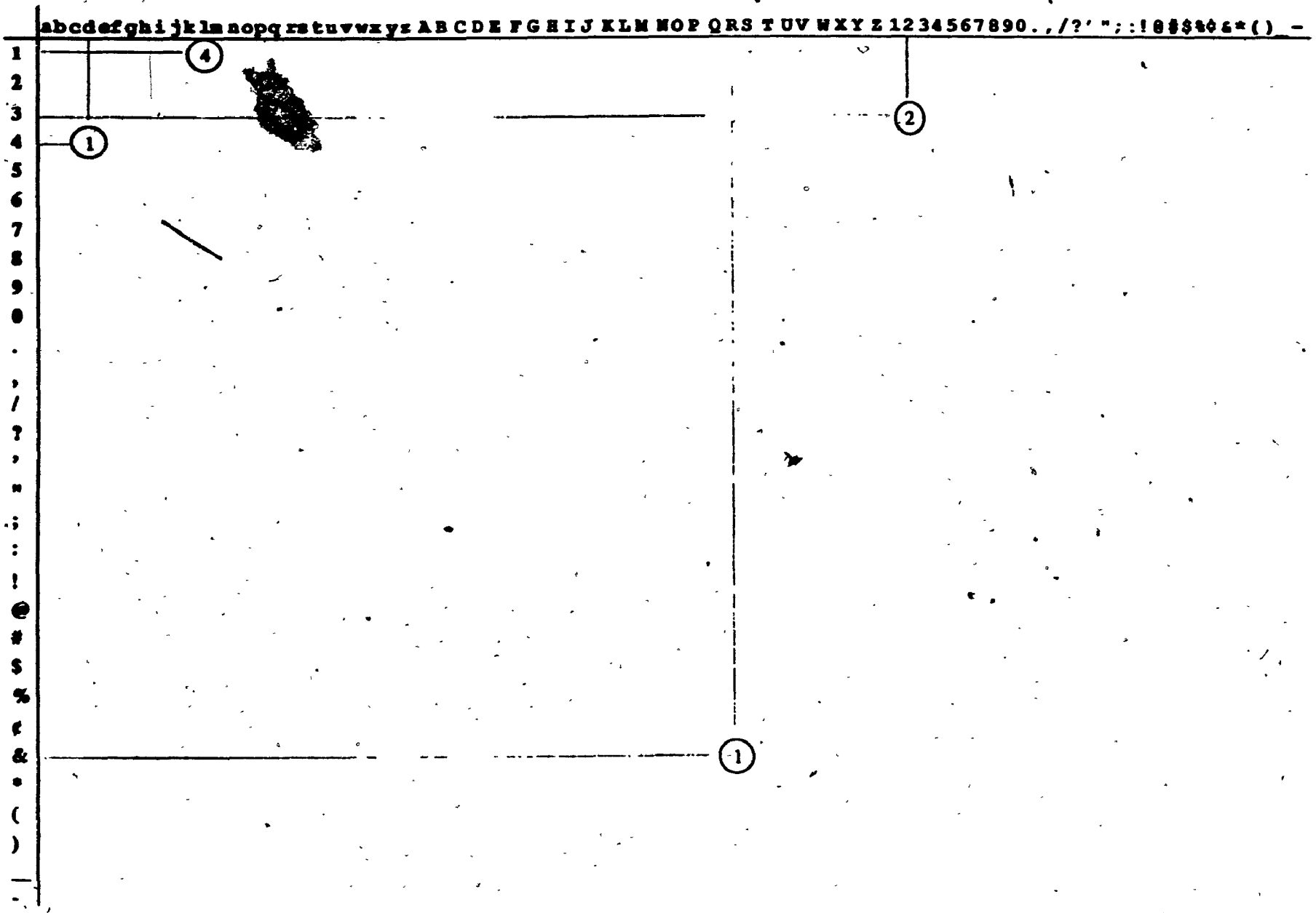
3.3.1 CONFUSION TABLE - PICA



CONFUSION TABLE - PICA (cont'd)



CONFUSION TABLE - PICA (cont'd)



3.4 Courier Font Results

Recognition Results

#chars tested	# data sets	# correct	# of errors	# rejected	% success	average time (sec/ dataset)*	average time (sec/ char)*
1577	19	1552	25**	0	98.4**	129.92 +/- 2.3	1.57 +/- .03

* *includes time it takes to read in data, scale it and provide results.*

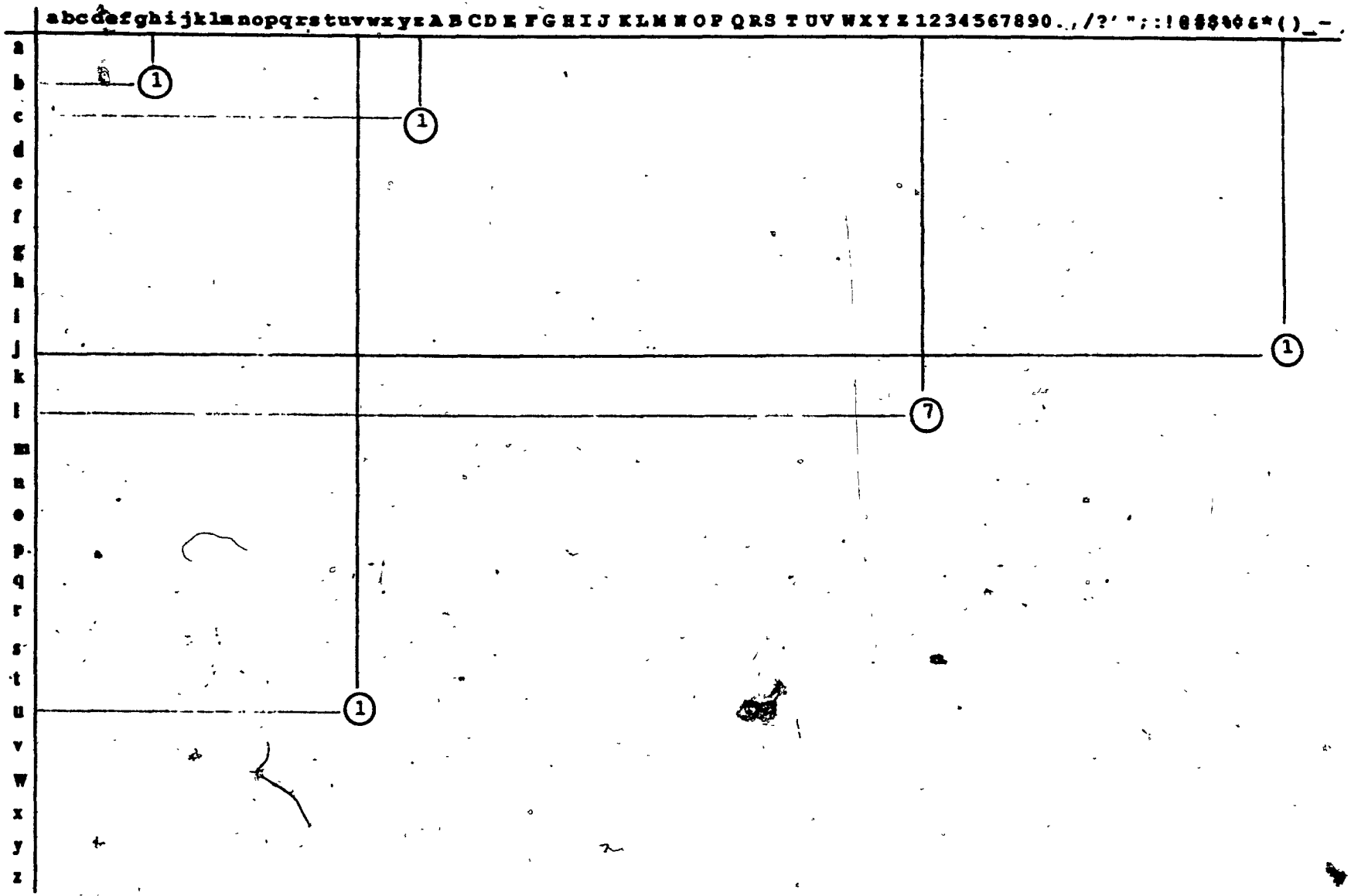
** *errors calculated with the assumption that the characters '1' and 'l' are indistinguishable in the Courier font. If misrecognitions between these two characters are included the number of errors reaches 43 and the percent success becomes 97.3%*

Timing Results (seconds per character)

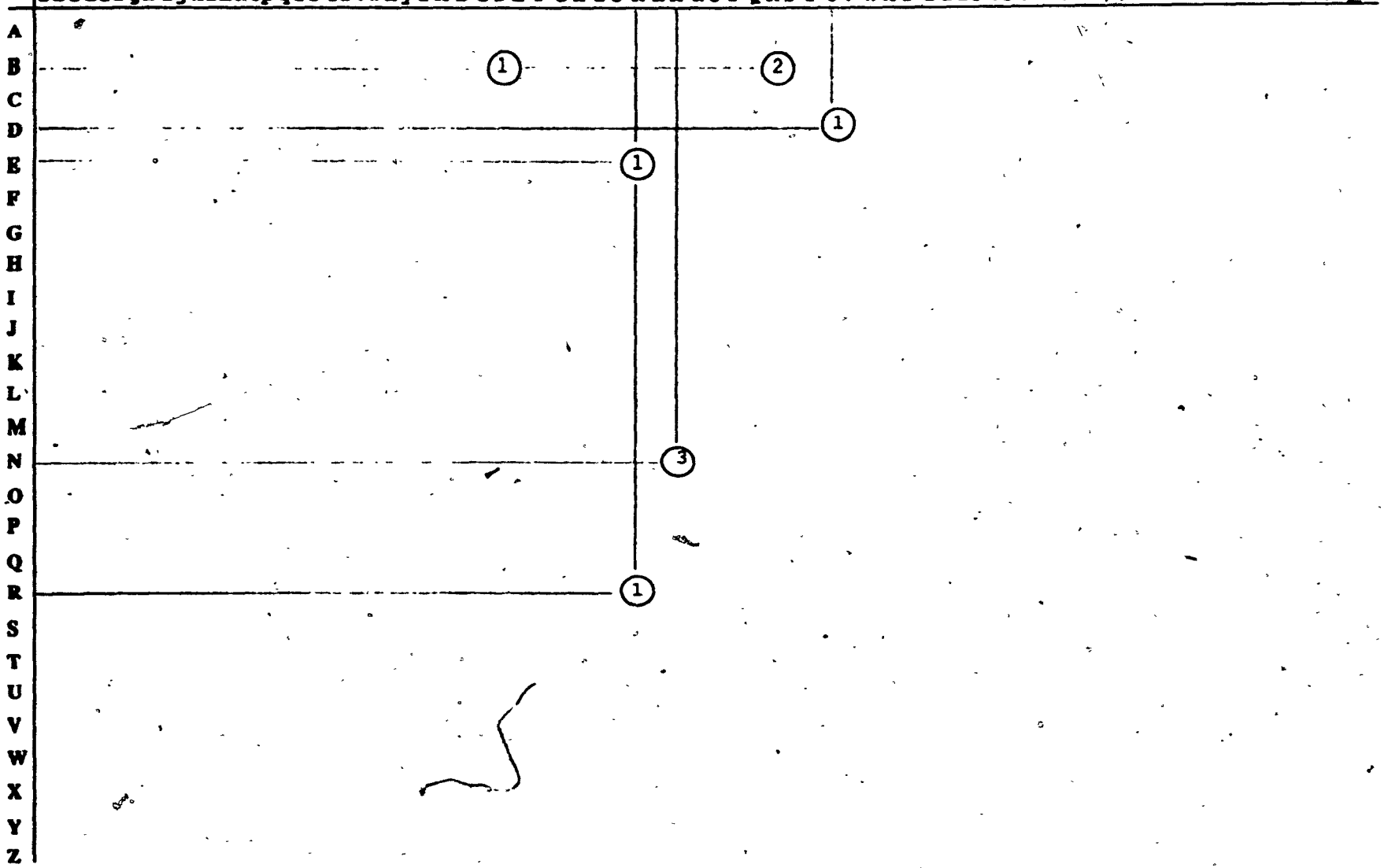
total time*	system overhead (reading data,	thinning, filling	decision making
1.57 +/- .03	1.01 +/- .0012	0.35 +/- 0.25	0.21 +/- 0.018

* *includes time it takes to read in data, scale it and provide results.*

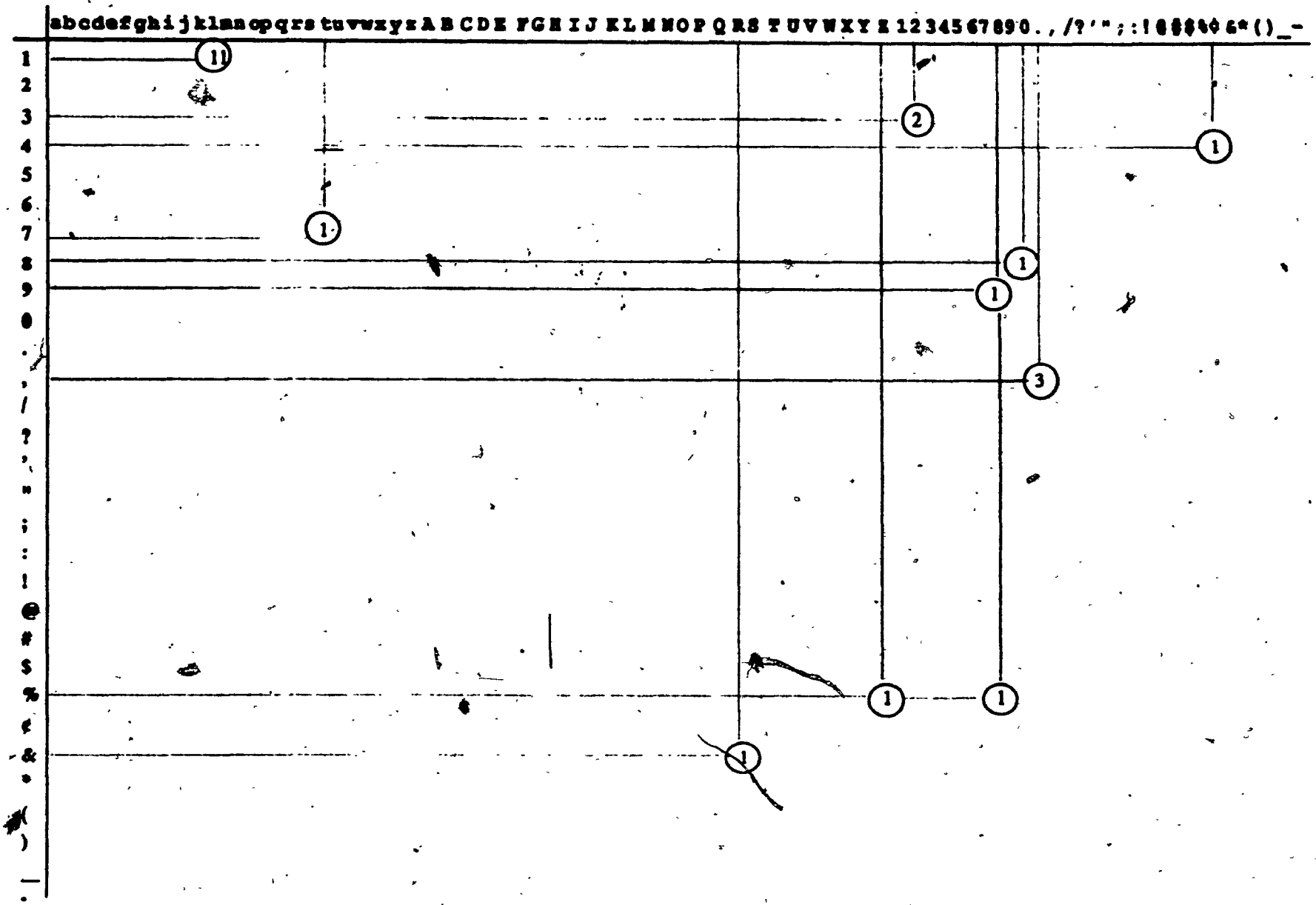
3.4.1 CONFUSION TABLE - COURIER



CONFUSION TABLE - COURIER (cont'd)



CONFUSION TABLE - COURIER (cont'd)



4.0 Discussion

4.1 Means of Improving Performance

4.1.1 Optimization in Preprocessing

The preprocessing of input data involves the filling and thinning of the data to remove any extraneous noise. The filling procedure eliminates any stray white spaces that should be dark (i.e. holes or gaps in the data) - an example being a white space completely surrounded by black pixels. Similarly, thinning eliminates any stray noise around the edges of a character.

Both procedures entail a similar approach. Each pixel in the data is analyzed with respect to all its surrounding pixels (a 3x3 window). A decision is then made whether or not this central pixel should be white or black.

The equations used to make this decision are the following (the pixels refer to the figure below):

(EQN. 1)

- thinning: X' = X AND (((A OR B OR D) AND (E OR G OR H)) OR ((B OR C OR E) AND (D OR F OR G)))

(EQN. 2)

- filling: X' = X OR (((B AND G) AND (D OR E)) OR ((D AND E) AND (B OR G)))

(X' being the new value of X)

A	B	C
D	X	E
F	G	H

Given a matrix of 40x40 pixels, each of the above equations must be calculated a total of 1600 times. Given the complexity of the equation, the time consumption is significant. In order to speed up the preprocessing, the following shortcut was implemented. Given that a character variable in the C language on the IBM PC is 1 byte, and each pixel analyzed in a 3x3 grid has eight neighbors, it is possible to map each neighbor pixel to one bit in the byte. Thus each possible combination of the neighbors can be accounted for by a unique, 1 byte number.

A short routine was written for each of the filling and thinning equations that worked out the results of (((A OR B OR D) AND (E OR G OR H)) OR ((B OR C OR E) AND (D OR F OR G))) (EQN 1), and (((B AND G) AND (D OR E)) OR ((D AND E) AND (B OR G))) (EQN 2) for all possible neighbor combinations. These tables were stored in separate files and read into core memory when the character recognition system was started up.

When preprocessing is to be carried out, the values of the pixels need only be assigned to their respective bits in a variable byte and all that has to be done to solve the equation is to index into the appropriate table with that byte. Thus the need to perform the complex part of the equations is simply transformed into an index into an array. An index to an array is, of course, faster than needed to process the complicated conditional statements. The only drawback is the fact that more memory is consumed by the program. However, the memory requirements of these two tables are only 256 bytes each for a total of 512 bytes. This is not a particularly heavy additional load.

This modification improved the speed of preprocessing by approximately 3 fold.

A further improvement in performance was achieved by only preprocessing the data where character data is actually present, i.e. skipping those rows which contain only white space. (The first and last rows with black space are calculated when the data is first read in - see flow chart).

4.1.2 Data Storage

In order to optimize data access, all data blocks (where possible) were stored on word boundaries. This simplifies matters in computing addresses. A significant example of this technique, was the storage of the input data itself. The data was provided in a 40x40 matrix. Instead of storing the data in an array of this size, the data was stored in an array of 64x64. Thus when a specific pixel in the array was to be accessed, a pointer was used and this pointer was computed within the program using simple additions and logical shifts. This saves the compiler from producing code with costly (in terms of time) multiplications to compute the needed address. If one starts with a pointer pointing to the DATA[0][0] then any array element DATA[i][j] may be computed as follows:

given that

```
*pt_DATA = DATA[0][0]
DATA[i][j] = *(pt_DATA + i + (j<<6))
DATA[3][0] = *(pt_DATA + 3)
DATA[0][5] = *(pt_DATA + (5 << 6))
```

note: the shift is 6 bytes because the size of the second dimension of the array is 64 = 26

a << b is the C Language syntax for a bitwise left shift of the argument a, b bytes.

```
DATA[3][5] = *(pt_DATA + 3 + (5<<6))
```

An example of how this technique is used is provided in the code for the preprocessing of the image.

4.1.3 Pattern Matching

A simple pattern matching algorithm was used to find the difference between the crossing pattern found for each character and those stored for the characters in the decision matrix determined by the physical properties of the input character. It consisted of simply subtracting a stored matrix from the one being tested. The best match yielded the result. If the difference between the stored crossing pattern of the determined result and the input character was above a given threshold, the character was rejected.

Optimization of the pattern matching routine - i.e. the routine that compares the crossing pattern to the input data against those stored in memory - required coding the routine in Assembly language. Since this routine is the most heavily used in the recognition process and is the most often repeated, coding it in assembler was the best way to insure optimal speed.

To more clearly understand how the pattern matching is performed, let us analyze the case of the character "'" (single quote, Gothic font). The first step is to determine the crossing numbers of the character. The input yielded the following:

horizontal crossing values: {1, 1, 1, 1, 1, 1, 1, 1, 1}

vertical crossing values: {1, 1, 1, 1}

Since the height of the character was 9, testing continued on the "very short" branch of the decision tree. (See section 2.5.1.1). The pattern matching process must match these crossing patterns against those of the characters termed as "very short". These include the characters:

"(double quote), '(single quote), _(underscore), -(dash), ,(comma).

As stated above, pattern matching is accomplished by finding the difference between the input character's crossing patterns and those of the characters it is being matched against.

In the case described above, the crossing numbers maintained by the system for the double quote are:

horizontal crossing values: {2, 2, 2, 2, 2, 2, 2, 2, 1}

vertical crossing values: {1, 1, 1, 1, 0, 0, 1, 1, 1}

The difference in the crossing numbers is determined as follows:

(horizontal)		
":	{2, 2, 2, 2, 2, 2, 2, 2, 1}	
input character:	{1, 1, 1, 1, 1, 1, 1, 1, 1}	
difference	1+1+1+1+1+1+1+1+1	= 8
(vertical)		
":	{1, 1, 1, 1, 0, 0, 1, 1, 1	
input character:	{1, 1, 1, 1}	
difference	0+0+0+0+0+0+1+1+1	= 3
total difference		= 11

This process is repeated for the other characters in the "very short" branch. The results for all the characters are tabled below.

character tested	net difference
"	11
'	1
_	28
-	12
,	5

The results lead to the recognition of the character as a single quote.(')

4.2 Limiting Factors

Limiting factors in terms of improving performance can be categorized in three main groups:

A. The nature of the IBM PC and MS/DOS. It is NOT a multiprocessing machine. A system such as the one described above would lend itself quite well to an operating system that allowed multiple processes running in parallel. An obvious advantage of such a machine would be in the ability to run various pattern matching tasks simultaneously and thus cut down the decision making time drastically.

B. The crossing algorithm itself. Due to the nature of the algorithm, preprocessing of the characters is necessary. Any stray pixel would throw off the crossing numbers quite substantially. Since the differences in crossing numbers between many characters is not very high, such minor noise would likely produce misrecognitions. Thus preprocessing is necessary to remove any such noise.

C. The nature of the input data. The input data in the form of a 40x40 ASCII matrix provides quite a bit of information for the system to process. Unfortunately, for the same reason described above (point B), it is impractical to shrink the data to a smaller matrix. There would not be sufficient information left to allow the crossing method algorithm to provide meaningful results.

5.0 Concluding Remarks

The results of this project lead one to several clear conclusions. The recognition of typed characters on an IBM PC can be accomplished with relatively little expense both in time and equipment. A full page of data should be able to be processed in well under 4 minutes. (Assuming 250 words/page and approximately 5 letters per word, as well as speed improvement running a true IBM PC rather than a clone) For household use this could be considered as acceptable. A recognition rate of close to 99% is also acceptable for such use. However, for commercial or industrial use, these speeds would be prohibitively slow.

To provide a potential user with a truly useful system, there must be a more efficient manner of passing of data from the scanner to the recognition process. The means utilized in this project are obviously not viable for a true system, nor was it meant to be. The concept here was to provide an efficient recognition system. The data flow into the actual recognition system was not considered as a part of this project.

The use of four different fonts gives one some further insight into the pros and cons of using any particular font for character recognition. Each font had its own idiosyncrasies, its own sets of characters that were particularly hard to distinguish between. Furthermore, each font had characters that were indistinguishable (for example, 'l' and '1' in pica). In a regular typewritten letter, a human reader can distinguish between these fonts by the context of the character within a word (word recognition, if you will) or the word within a sentence. It is interesting to note that although a context-free approach would be the ideal approach to character recognition systems, it is recognized by researchers in the field that "a good recognition system cannot be truly context free....in classifying a character, it is very helpful to see what its predecessor was...".[2]. Since the characters tested were not provided in the form of words, just strings of characters, such contextual information was not available.

A system designed for simplicity and low cost does not have the ability to provide the service of more elaborate word recognition. A more sophisticated system may be able to attach a dictionary and provide spell checking and correction to eliminate such problems. Such added features, would of course, impact the system with added cost and increased processing time. It is unlikely that an 8088-based system could cope well with such a load. Thus, in a system such as the one described above, these types of errors must be accepted and 'lived with' if dealing with these fonts. A positive alternative would be to provide recognition of the OCR fonts (OCR A, OCR B). However, these fonts are not commonly used. This would then severely limit the practicality of this system.

REFERENCES

1. **Ahmed, P.** *Computer Recognition of Totally Unconstrained Handwritten ZIP codes.* Dept. Computer Science, Concordia University, Montreal. July, 1986.

2. **Caskey, D.L., C.L. Coates.** *Machine Recognition of Handprinted Characters.* First Int'l Joint Conf. Pattern Recognition, Washington, 1973, pp. 41-49.

3. **Duda, R. O., P. E. Hart.** *Pattern Classification and Scene Analysis.* John Wiley & Sons. N.Y. 1973.

4. **Hilditch, C. J.** *Linear Skeletons form Square Cupboards. In Machine Intelligence Vol. IV, . eds: B. Meltzer and D. Michie.* Edinburgh University Press, Edinburgh, 1969, pp. 403-420.

5. **Kernighan, B. W., D. M. Ritchie.** *The C Programming Language.* Prentice Hall, 1978.

6. **Narasimhan, R.** *On the Description, Generation and Recognition of Classes of Pictures. In Automatic Interpretation and Classification of Images, ed. A. Grasselli.* Academic Press. N.Y. 1969, pp 1-42.

7. **Stanton, T., D. Burns, S. Venit.** *Page to Disk Technology: Nine State-of-the-Art Scanners.* PC Magazine V5(16) 1986, pp. 128-177.

8. **Suen, C. Y.** *Distinctive Features in Automatic Recognition of Handprinted Characters.* Signal Processing. V4(2-3) Apr. 1982. pp. 193-207.

9. **Suen, C. Y.** *Character Recognition by Computer and Applications. In Handbook of Pattern Recognition and Image Processing, eds. T.Y. Young and K.S. Fu.* Academic Press, Orlando, 1986, pp. 569-585.

10. **Tucker, N. D., F. C. Evans.** *A Two-Step Strategy for Character Recognition Using Geometrical Moments.* Int'l Joint Conf. Pattern Recognition, Copenhagen, 1974, pp. 223-225.

APPENDIX A

SAMPLES OF INPUT DATA (before scanning)

q w e r t y u i o p [¶

a s d f g h j k l ; ' >

§ z x c v b n m , . /

Q W E R T Y U I O P] †

A S D F G H J K L : " <

° Z X C V B N M , . ?

! @ # $ % ¢ & * ( ) _ +

1 2 3 4 5 6 7 8 9 0 - =

courier

Lowercase

q	w	e	r	t	y	u	i	o	p	[	¶
a	s	d	f	g	h	j	k	l	;	'	>
§	z	x	c	v	b	n	m	,	-	/	

UPPERCASE

O	W	E	R	T	Y	U	I	O	P	]	†
A	S	D	F	G	H	J	K	L	±	"	<
°	Z	X	C	V	B	N	M	,	-	?	

Numeric characters

1	2	3	4	5	6	7	8	9	0	—	=

Special characters

!	@	#	$	%	¢	&	*	(	)	_	+

Lowercase

q w e r t y u i o p [¶

a s d f g h j k l ; ' >

§ z x c v b n m , . /

UPPERCASE

Q W E R T Y U I O P] †

A S D F G H J K L : " <

° Z X C V B N M , . ?

Numeric characters

1 2 3 4 5 6 7 8 9 0 - =

Special characters

! @ # $ % ¢ & * ( ) _ +

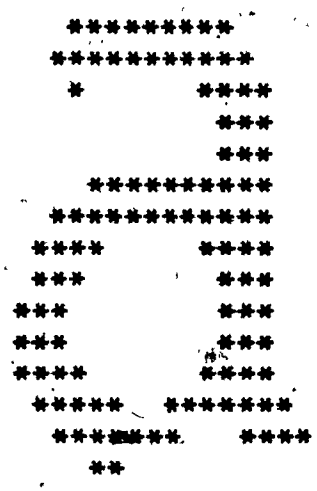
gothic

```
q  w  e  r  t  y  u  i  o  p  [  ¶
a  s  d  f  g  h  j  k  l  ;  '  >
§  z  x  c  v  b  n  m  ,  .  /

Q  W  E  R  T  Y  U  I  O  P  ]  †
A  S  D  F  G  H  J  K  L  :  "  <
°  Z  X  C  V  B  N  M  ,  .  ?

!  @  #  $  %  ¢  &  *  (  )  _  +

1  2  3  4  5  6  7  8  9  0  -  =
```

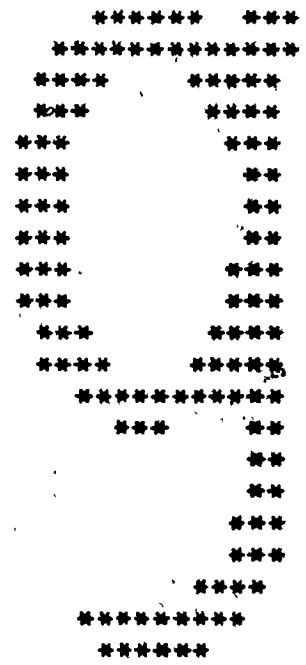
pica

APPENDIX B

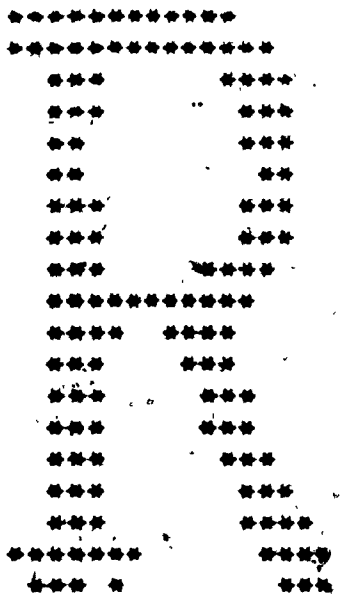
SAMPLES OF INPUT DATA (after scanning)



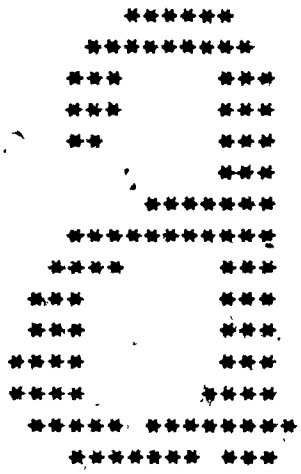
COURIER - letter 'a'



COURIER - letter 'g'



COURIER - letter 'R'



PICA - letter 'a'

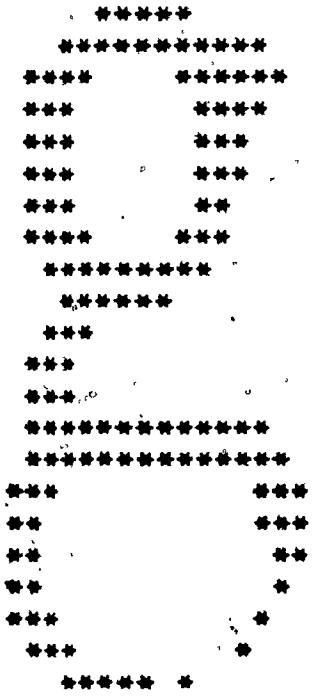
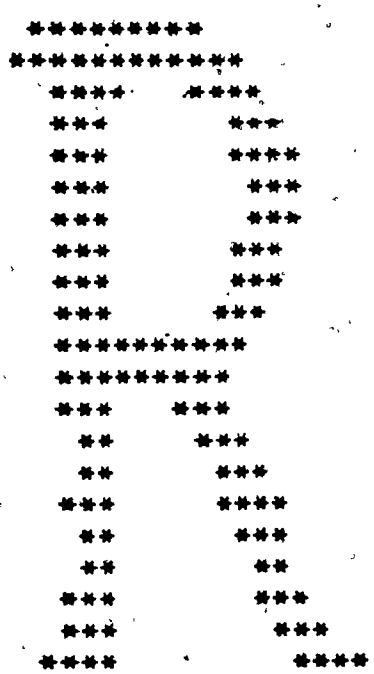
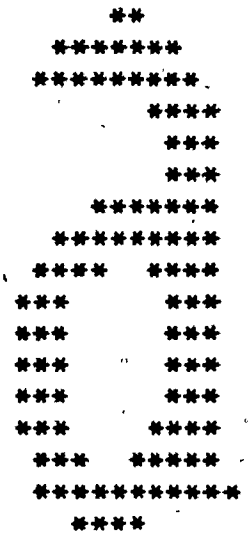


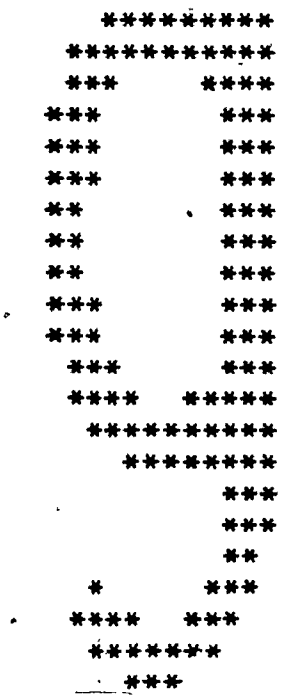
FIG - letter 'g'



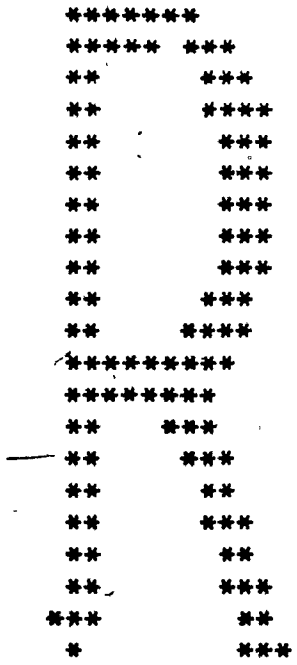
PICA - letter 'R'



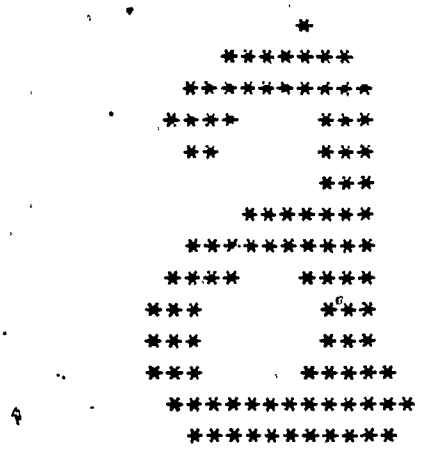
GOTHIC - letter 'a'



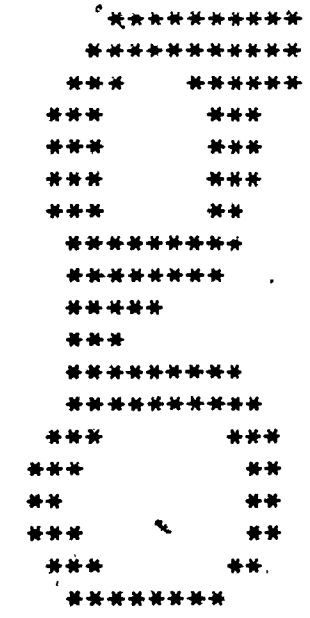
GOTHIC - letter 'g'



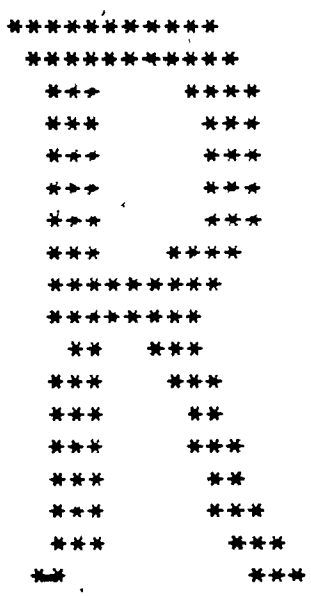
GOTHIC - letter 'R'



ELITE - letter 'a'



ELITE - letter 'g'

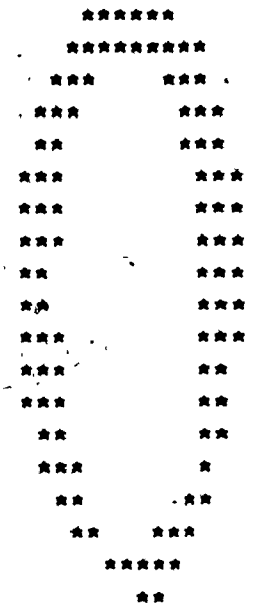


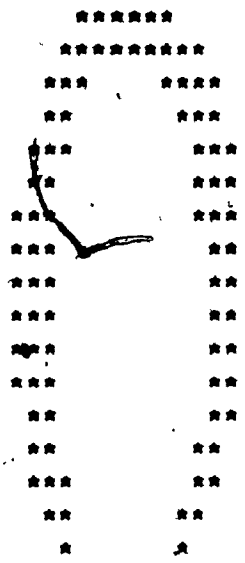
ELITE - letter 'R'

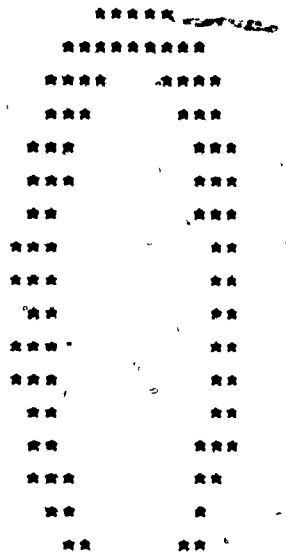
APPENDIX C

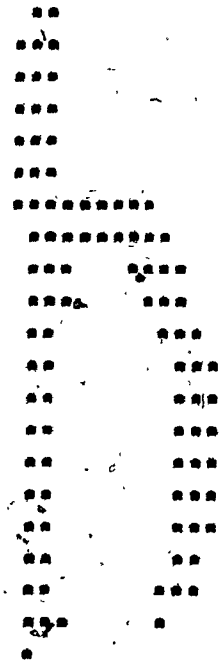
SAMPLES OF ELITE DATA

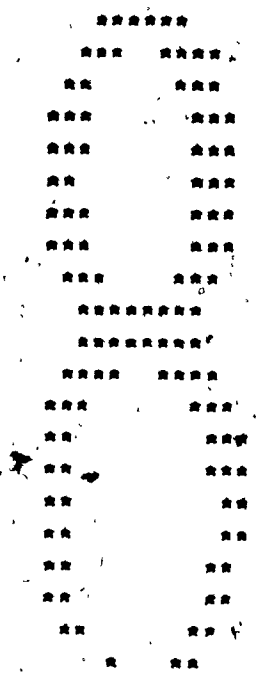
A> -











APPENDIX D

SAMPLES OF GOTHIC DATA