[12] K. Fukunaga, *Introduction to statistical pattern recognition*, Academic press, 1972

[13] P.J. Verveer and R.P.W Duin, "Estimators for the intrinsic dimensionality evaluated and applied," Delft University of Technology, Faculty of Applied Physics, Pattern Recognition Group, Technical Report, 1993

[14] Ph.A. Errington and J. Graham, "Application of artificial neural networks to chromosome classification," *Cytometry*, vol. 14, pp. 627–639, 1993

Automated Evaluation of OCR Zoning

Junichi Kanai, Stephen V. Rice,
Thomas A. Nartker and George Nagy

Abstract— **Many current optical character recognition (OCR) systems attempt to decompose printed pages into a set of zones, each containing a single column of text, before converting the characters into coded form. We present a methodology for automatically assessing the accuracy of such decompositions, and demonstrate its use in evaluating six OCR systems.**

Index Items—**Document image understanding, page segmentation, layout analysis, performance evaluation metric.**

I. Introduction

The first step in Optical Character Recognition (OCR) is to locate and order the text to be recognized. Commercial OCR systems allow the user to demarcate the text regions on a page image by drawing boundaries around them. The order of these regions, or zones, is significant. It normally corresponds to the reading order of the page. This process is known as *manual zoning*.

Alternatively, the user can let the OCR system automatically identify text regions and their order. The system finds columns of text and, if they are not part of a table, defines a separate zone for each column so that the generated text will be "de-columnized." This is also known as "galley format." In addition, the system identifies graphic regions in order to exclude them.

Zone representation schemes are not standardized. The following methods are used by commercial OCR systems: bounding rectangles, piecewise rectangles, polygons, and nested rectangles (see Fig. 1.). In some rare instances, zones may overlap. Moreover, deskewing the page alters the zoning. Therefore, geometric comparison of zones is not feasible. A recently proposed alternative to our method is the comparison of the configuration of black pixel sets, but this method precludes testing "black-box" commercial systems [1].

In order to evaluate the accuracy of automatic zoning, we introduce a *zoning metric* based on the number of edit operations required to transform an OCR output to the correct text. The string of characters generated by the system under evaluation on an unzoned page image is compared to the correct text string. All mismatched substrings are identified by a divide-and-conquer string matching algorithm that finds the longest matches first. Then the number of editing operations (deletions, insertions, and moves) required to correct the system output is computed. Finally, the number of editing operations required to correct the output of the same OCR system operating on a manually-zoned version of the page image is subtracted to yield the cost of correcting only zoning errors. All of the above operations, with the exception of producing the "true" string for each page, are performed automatically.

Section II presents the move-counting and string-matching algorithms in greater detail. Section III describes the test data and the experimental protocol. The results obtained by processing 460 printed pages on six commercial systems are discussed in Section IV. Section V points to future research.

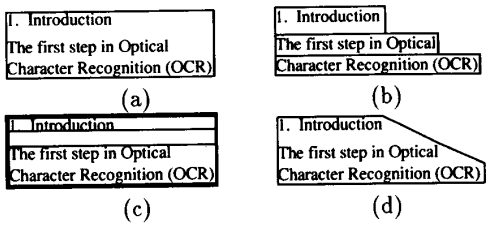

Fig. 1. Zone Representation Schemes. (a) bounding rectangle, (b) piecewise rectangles, (c) nested rectangles, (d) polygon.

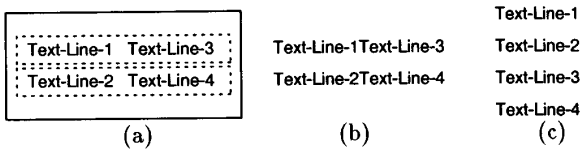

Fig. 2. Example of Zoning Error. (a) incorrectly zoned page, (b) generated text, (c) correct text.

II. Zoning Metric

A human editor utilizes three kinds of operations to correct OCR-generated text: *insertion, deletion,* and *move.* If a zoning algorithm misclassifies a text region as a graphic region or does not detect a text region, the characters in the text region are not recognized by the OCR algorithm. Thus, the editor must insert (type) the missing characters into the OCR output.

On the other hand, if a graphic region is misclassified as a text region, characters in the graphic region are included in the OCR output. Furthermore, graphic objects could be converted into a set of characters. For example, the vertical axis of a graph might become I's. Such phantom characters must be deleted from the OCR output.

When a multi-column page is incorrectly zoned as shown in Fig. 2., Text-Line-3 must be moved between Text-Line-2 and Text-Line-4. Therefore, the cost of correcting the reading order of an OCR output is an important part of measuring the performance of a zoning algorithm.

A move operation can be performed by either *cut and paste* or *delete and re-type.* The human editor will normally make use of a *cut and paste* capability to move a string of n characters to its correct location. But for n less than some threshold T, it is easier (and less costly) to perform n deletions and n insertions to make the correction. The value of T will vary depending on the skills of the human editor and the editing tools at hand, but is most likely to be in the range of 5 to 100.

We propose a two-part algorithm for calculating the cost of correcting the OCR-generated text. First, the minimum number of edit operations is estimated (see Fig. 3.). Next, the total cost is calculated

Manuscript received July 9, 1993; Revised July 7, 1994. Recommended for acceptance by Dr. Rangachar Kasturi.

J. Kanai, S. V. Rice, and T. A. Nartker are with the Information Science Research Institute at UNLV, Las Vegas, NV 89154-4021, USA.

G. Nagy is with the ECSE Department, Rensselaer Polytechnic Institute, Troy, NY 12180-3590, USA.

IEEE Log Number P95007

according to the cost of each operation.

Let S_c be a string of characters corresponding to the correct text of a page and S_o be an OCR output using the automatic zoning option. S_c and S_o are compared, and matches (common substrings), including transposed matches, are identified (see Section II A for details). The number of unmatched characters D in S_o corresponds to the number of unnecessary characters generated by the OCR system. Therefore, D is the number of deletion operations needed.

Using an algorithm described in Section II B, the minimum number of moves M required to rearrange the matches of S_o in the proper order is calculated.

```
begin
   compare S_c with S_o and identify matches using the
   algorithm described in Section II A (Fig. 4.)
   D ← number of unmatched characters in S_o
   M ← number of moves computed by the algorithm
        described in Section II B (Fig. 5)
   I ← number of unmatched characters in S_c
   for each move of length L < T do
      I ← I + L
      D ← D + L
      M ← M − 1
   end for
end
```

Fig. 3. Algorithm for Counting Edit Operations. Given strings S_c and S_o and threshold T, the algorithm yields the number of insertions, I, the number of deletions, D, and the number of move operations, M.

The number of unmatched characters I in S_c corresponds to the number of characters that were dropped by the OCR system. Thus, I is the number of insertion operations needed.

```
Given strings S_c[1: n_c] and S_o[1: n_o]
  begin
     Φ(S_c) ← {[1: n_c]}
     Φ(S_o) ← {[1: n_o]}
     loop
        find [α_c : ω_c] ∈ Φ(S_c), [α_o : ω_o] ∈ Φ(S_o),
        and α_c', ω_c', α_o', ω_o', where α_c ≤ α_c' ≤ ω_c' ≤ ω_c
        and α_o ≤ α_o' ≤ ω_o' ≤ ω_o, such that
        S_c[α_c' : ω_c'] = S_o[α_o' : ω_o'] is the longest substring
        common to an unmatched substring of S_c and S_o
        if not found then
           exit
        output the match described by α_c', ω_c', α_o', and ω_o'
        Φ(S_c) ← Φ(S_c) ∪ {[α_c : α_c'−1],[ω_c'+1: ω_c]}−{[α_c : ω_c]}
        Φ(S_o) ← Φ(S_o) ∪ {[α_o : α_o'−1],[ω_o'+1:ω_o]}−{[α_o: ω_o]}
     end loop
  end
```

Fig. 4. String-Matching Algorithm.

Each move operation of length L less than the threshold T is converted to L insertions and L deletions so that the short moves are performed by deleting and re-typing. Thus, the number of moves M is reduced. The total number of insertions and deletions which result from this conversion are determined.

In the second step, the following cost model is used to calculate the total editing cost. We assume that the cost of a move operation is independent of the distance moved and that all moves of strings of length L greater than T have the same cost. Let W_i, W_d, and W_m be the costs associated with an insertion, a deletion, and a move, respectively. Since the user moves rather than deletes and re-types a character string when it is cheaper, T should be equal to $W_m / (W_i + W_d)$. The total cost of correcting the OCR-generated string is calculated by the following formula:

$$\begin{aligned} &Cost(S_o, S_c, W_i, W_d, W_m, T) \\ &\quad = W_i{*}I + W_d{*}D + W_m{*}M \\ &\quad = W_i{*}I + W_d{*}D + (W_i + W_d){*}T{*}M \\ &\quad = Cost(S_o, S_c, W_i, W_d, T) \end{aligned}$$

This cost model charges a cost W_i for each insertion, and W_d for each deletion made by the OCR process. A substitution error is corrected by deleting an incorrect character and inserting the correct character; therefore, it costs $W_d + W_i$.

OCR output could contain errors made not only by the automatic zoning process but also by the character recognition process. *Cost* is the total cost for correcting all types of errors. To eliminate the effects of recognition errors, a calibrated cost must be calculated. We assume that OCR systems make the same recognition errors when a page is manually zoned and automatically zoned. Let S_m be the character string generated from a manually zoned page. The cost of transforming S_m into the correct text S_c results from correcting recognition errors. The calibrated cost is defined by:

$$\begin{aligned} &Calibrated\_Cost(S_o, S_m, S_c, W_i, W_d, T) \\ &\quad = Cost(S_o, S_c, W_i, W_d, T) - Cost(S_m, S_c, W_i, W_d, T) \end{aligned}$$

This calibrated cost effectively isolates the cost of zoning the page automatically. It should be compared with the cost of zoning the same page manually.

A. String Matching

String matching has been studied in detail, and some applications in OCR were described in [2]. In our algorithm (shown in Fig. 4.), the correct string, S_c, is matched with the generated string, S_o. This algorithm is similar to one presented in [3], but since S_o may contain blocks of text in the wrong order, this algorithm must detect transposed matches.

Let S be a string of n characters ($n \geq 0$). Let $S[\alpha : \omega]$ denote the substring of S containing the α^{th} through ω^{th} characters inclusive. Normally $\alpha \leq \omega$, but if $\alpha > \omega$, then $S[\alpha : \omega]$ represents the null (zero-length) string. S is equivalently represented by $S[1 : n]$.

Let $\Phi(S)$ denote the set of unmatched substrings of S. That is,

$$\Phi(S) = \{[\alpha : \omega] \mid S[\alpha : \omega] \textit{ is an unmatched substring}\}.$$

Initially, all characters are considered unmatched. The longest substring common to S_c and S_o is found and constitutes the first match. $\Phi(S_c)$ and $\Phi(S_o)$ are then updated to exclude this substring. This results in two unmatched substrings for each string: the characters preceding the match, and the characters following it.

The second match is determined by finding the longest substring common to an unmatched substring of S_c and S_o. When this match is excluded, there are three unmatched substrings for each string. This process of examining the unmatched substrings continues until no common substring can be found, i.e., S_c and S_o have no unmatched characters in common.

B. Move-Counting

Fig. 5. shows the algorithm used for counting move operations based on Latifi's method [4]. The N matched substrings of the generated text are represented by a permutation of the integers 1 to N. For example, given the correct string,

$$\overbrace{\textit{the quick}}^{1}\ \overbrace{\textit{red}}^{2}\ \overbrace{\textit{fox}}^{3}\ \overbrace{\textit{jumped over the}}^{4}\ \overbrace{\textit{lazy dog}}^{5}\ ,$$

and the generated string,

$$\overbrace{\textit{the quick}}^{1}\ \overbrace{\textit{jumped over the}}^{4}\ \overbrace{\textit{fox}}^{3}\ \overbrace{\textit{lazy dog}}^{5}\ \overbrace{\textit{red}}^{2}\ ,$$

there are five matches and the permutation is (1 4 3 5 2).

Adjacent substrings are combined so that the permutation has no two consecutive integers, ($\cdots$ x [x+1] $\cdots$). For example, the permutation (5 2 3 1 4) is re-written as (4 2 1 3).

This algorithm determines the number of move operations required to correct the generated text by counting the number of moves needed to transform this permutation into the identity permutation, i.e., (1 2 $\cdots$ N), or re-written as (1).

At each step, the algorithm attempts to find a move operation that yields the greatest reduction in the size of the permutation (after re-writing). Latifi has shown that the largest such reduction is three; this occurs when the permutation contains a sequence of the form, ($\cdots$ y x [y+1] $\cdots$), and a sequence of the form, ($\cdots$ [x–1] [x+1] $\cdots$), and x is moved between [x–1] and [x+1]. For example, given the permutation (5 1 3 2 4), a reduction of three is achieved by moving 3 after 2; the permutation becomes (5 1 2 3 4), or re-written as (2 1).

If there is no move available that yields a reduction of three, the algorithm looks for a move that results in a reduction of two. If the permutation contains either a sequence of the form, ($\cdots$ y x [y+1] $\cdots$), or a sequence of the form, ($\cdots$ [x–1] [x+1] $\cdots$), then x can be moved after [x–1], and/or before [x+1], to achieve a reduction of two.

Finally, if there is no move available that yields a reduction of two or three, then any x can be moved after [x–1], or before [x+1], to obtain a reduction of one.

Whenever there is more than one move operation available yielding the same reduction, the one moving the fewest number of characters is chosen. This models the behavior of the human editor more accurately, and results in more move operations falling below the threshold T.

```
Given permutation P:
  begin
    count ← 0
    while P is not the identity permutation do
      if P can be reduced by three then
        reduce P by three
      else if P can be reduced by two then
        reduce P by two
      else
        reduce P by one
      count ← count + 1
    end while
  end
```

Fig. 5. Move-Counting Algorithm.

The number of move operations is tightly bounded by $N/3$ and N [4]; on average, the algorithm finds $N/2$ moves. In general, humans have difficulty determining the minimum number of move operations, and since we are modeling human editors, an optimal move-counting algorithm is not necessary. Furthermore, an optimal algorithm will likely be computationally expensive.

Empirical data show that N can be as high as 750. On average, for a typical page of 1,800 characters, N is 47, and the number of moves counted by this algorithm is 27, where 10 are reductions of one, 15 are reductions of two, and 2 are reductions of three.

III. Experiments

The automatic zoning capabilities of commercial OCR systems were studied using this zoning metric. Each system was treated as a black box and operated entirely automatically using the software tools described in [5]. Every system processed the same set of digitized pages.

A. Test Data and Systems

A set of 460 pages randomly selected from a set of 2,500 documents containing a total of 100,000 pages was used in this experiment. (See [6] for details.) Sample pages were digitized at 300 dpi using a Fujitsu M3096E+ scanner. The default threshold setting was used to produce the binary image of a page. Each page was manually zoned, and correct text was keyed in and verified corresponding to each zone. We excluded equations and text in graphic objects.

Although most pages have unique reading orders, for some pages, the reading order is not clear. There may be several equally-correct reading orders. Consider a figure placed in the middle of a page. Readers may or may not read the figure caption until they finish reading the main body text on the page.

There are two ways to deal with this problem. The first approach uses m character strings corresponding to the m equally-correct reading orders and calculates the minimum cost.

The second approach uses only a single correct reading order to calculate the cost. When zoning errors are made, many character strings corresponding to text-lines must be moved to correct the reading order. On the other hand, when an equally-correct reading order is selected, only a few long character strings corresponding to zones must be moved. In practice, choosing a different, but equally-correct, reading order has a negligible effect on the overall cost.

TABLE I
SUMMARY OF TEST DATA

Class	Pages	Characters
Single column	278	416,638
Table	107	144,106
Multi-column	75	257,202
Total	460	817,946

In this experiment, the second approach was taken. The single correct reading order was chosen according to the following rules.

1) No sentence on a page is divided by other text objects.
2) If a figure caption, table, or footnote interrupts the flow of main body text, it is placed after the main body text.

To study the behavior of these zoning algorithms in detail, the sample pages were divided into the following three classes according to their layout characteristics: *single column pages, table pages,* and *multi-column pages. Single column pages* consist of one column in which text flows. These pages may contain figures, indented blocks, or lists but do not contain any tables. *Table pages* contain at least one table and may also contain single column text. *Multi-column pages* have at least two columns and could contain a variety of objects; ten

pages contained small tables. Table I shows the number of pages and the number of characters in each class.

Six OCR systems (some not yet commercially available) from six different vendors were used in the experiment. Two of the systems run on Sun SPARCstations; the other systems run on PC's.

B. Modified Cost Model

The current generation of commercial OCR systems attempts to recognize all characters, including those in figures. In this experiment, the weight W_d is set to zero, so that OCR systems are not penalized for trying to recognize these characters.

$$\begin{aligned} &Cost(S_o, S_c, W_i, W_d, T) \\ &\quad = W_i*I + W_d*D + (W_i + W_d)*T*M \\ &\quad = W_i*I + W_i*T*M \\ &\quad = W_i*(I + T*M) \end{aligned}$$

Now the cost depends only on moves and insertions. Each move, however, is equivalent in cost to T insertions. Therefore, the overall cost of correcting zoning errors can be stated in terms of equivalent insertion operations.

IV. Results

The calibrated cost of correcting automatic zoning errors was calculated for each of the six systems (see [7] for details). Fig. 6. shows the cost of correcting automatic zoning errors as a function of T, i.e., the cost of a move in insertions. As the cost of a move operation increases, more character strings are deleted and re-typed rather than moved.

If moves are inexpensive relative to insertions, then the cost of correcting the 460 pages processed by the worst system is almost ten times higher than for the best system. However, as the relative cost of moves increases, the overall cost curves saturate because it pays to delete and retype, rather than move, even the longest strings. At an operator cost of \$10 per hour, and a rate of 2,000 insertions, or equivalently, 100 moves per hour (i.e., T = 20), correcting all the data processed by the best system would cost \$10 × 25,000 / 2,000 = \$125. This is the cost of correcting only the automatic zoning errors, which can be compared to the cost of manual zoning. Assuming it takes an average of two minutes to manually zone a page, then at the same labor rate, the 460 pages can be manually zoned for $\$10 \times 460 \times 2 / 60 = \$153$.

Fig. 7. shows the total costs for processing each class of pages by the best, middle, and worst OCR system; note that these costs are not normalized by the number of characters or pages in each class. The cost of correcting zoning errors on *multi-column pages* is higher than for *single-column pages* but not as high as the cost for *table pages*.

We examined pages to determine possible causes of zoning errors. As intuition suggests, these systems process *single column pages* well. We observed two types of errors that were made on these pages:

1) one or more missing paragraphs
2) moving some text to a wrong (usually adjacent) line.

It seems that this last type of error was made by a text-line extraction process. When text-lines are skewed or curved, some text near either end of a text-line may be incorrectly moved to an adjacent line. The zoning metric measures the cost of correcting this type of error and includes it in the total correction cost.

We observed two kinds of problems in de-columnizing *multi-column pages*. The first kind is related to the complexity of page layouts. Although these systems partition simple Manhattan page layouts well, they have problems with complex Manhattan page layouts, such as title pages in multi-column formats. The second kind of problem is related to inter-column spacing. When the space between columns is narrow and/or a page is skewed, some systems fail to de-columnize the page.

The most common error in *table pages* was de-columnization of tables. When tables are partially or completely de-columnized, many move operations are needed to reconstruct them. This observation suggests that most systems have difficulty distinguishing tables from multi-column text. Another problem was the dropping of some columns or parts of columns.

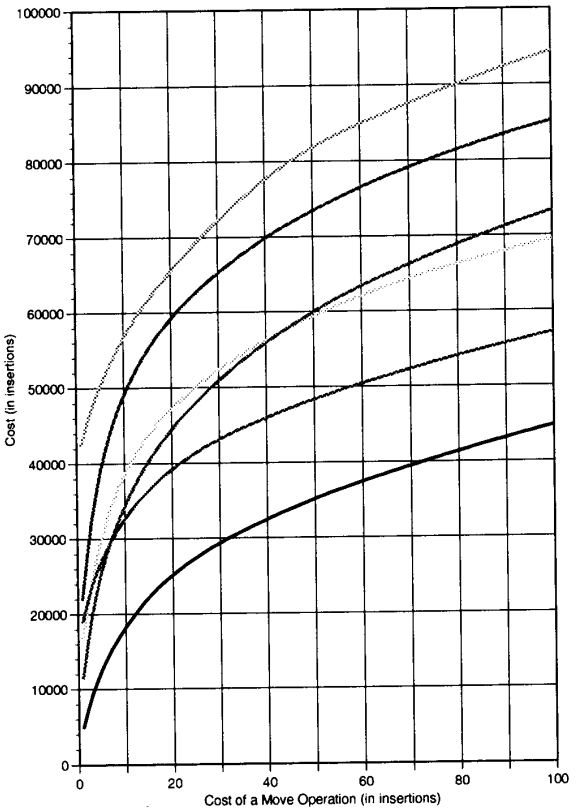

Fig. 6. Cost of Correcting Automatic Zoning Errors for six OCR systems.

V. Conclusions

We introduced an automated method for measuring the performance of zoning algorithms. This metric is based on an approximation of the cost of human editing to correct OCR-generated text. We presented two cost functions. The first function, *Cost*, calculates the cost of correcting all types of errors generated from automatically zoned pages. The second function, *Calibrated_Cost*, calculates only the cost of correcting the zoning errors.

Currently, this is the only known automated way to measure the performance of zoning algorithms that is independent of the zone representations used. This metric can be used to study the effects of image enhancement algorithms, such as deskewing algorithms, on automatic zoning.

Our preliminary results show that the current generation of zoning algorithms can process *single column pages* well. Yet, these algorithms have difficulties in processing pages containing tables. This

study suggests that the recognition of tables is an important research topic.

Finally, we note that the editing model could be extended to incorporate other edit operations, such as block deletion. In the experiment, we assigned the pages to one of three classes and studied the behavior of the automatic zoning algorithms. We plan to analyze further what kind of layout features make automatic zoning difficult and to study the skew sensitivity of these algorithms.

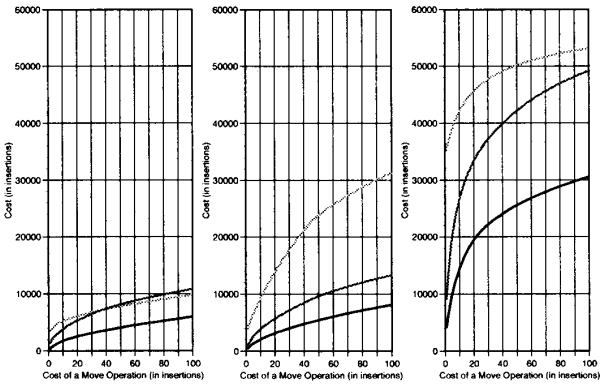

Fig. 7. Cost of Correcting Automatic Zoning Errors for the best, middle, and worst OCR system:(a)single column pages, (b) multi-column pages, (c) table pages.

ACKNOWLEDGMENT

This research was funded by the U.S. Department of Energy. We received valuable assistance from Professor S. Latifi (UNLV).

REFERENCES

[1] S. Randriamasy, L. Vincent, and B. Wittner, "An Automatic Benchmarking Scheme for Page Segmentation," *Document Recognition,* SPIE Proceedings Series, vol. 2181, pp. 217-230, 1994.

[2] S. N. Srihari, *Computer Text Recognition and Error Correction,* IEEE Computer Society, 1985.

[3] S. V. Rice, J. Kanai, and T. A. Nartker, "A Difference Algorithm for OCR-Generated Text," *Advances in Structural and Syntactic Pattern Recognition*, ed. H. Bunke, World Scientific, pp. 333-341, 1992.

[4] S. Latifi, "Correcting OCR-generated Text Using Permutations," *Proc. ICEE 93,* Amir Kabir University, Tehran, Iran, May 1993.

[5] S. V. Rice, "The OCR Experimental Environment, Version 3," University of Nevada, Las Vegas, Technical Report ISRI TR-93-04, April 1993.

[6] S. V. Rice, J. Kanai, and T. A. Nartker, "An Evaluation of OCR Accuracy," University of Nevada, Las Vegas, Technical Report ISRI TR-93-01, April 1993.

[7] J. Kanai, S. V. Rice, and T. A. Nartker, "A Preliminary Evaluation of Automatic Zoning," University of Nevada, Las Vegas, Technical Report ISRI TR-93-02, April 1993.

A Method of Combining Multiple Experts for the Recognition of Unconstrained Handwritten Numerals

Y. S. Huang and C. Y. Suen

***Abstract*—For pattern recognition, when a single classifier cannot provide a decision which is 100 percent correct, multiple classifiers should be able to achieve higher accuracy. This is because group decisions are generally better than any individual's. Based on this concept, a method called the "Behavior-Knowledge Space Method" was developed, which can aggregate the decisions obtained from individual classifiers and derive the best final decisions from the statistical point of view. Experiments on 46,451 samples of unconstrained handwritten numerals have shown that this method achieves very promising performances and outperforms voting, Bayesian, and Dempster-Shafer approaches.**

***Index Items*—Unconstrained handwriting recognition, combination of multiple classifiers, evidence aggregation, behavior-knowledge space, knowledge modeling.**

I. INTRODUCTION

The recognition of handwritten numerals has been studied for more than three decades; during this period, many classifiers with high recognition rates have been developed [1]. However, none of them can achieve satisfactory performance when dealing with characters of degraded quality. A new trend [2], [3], [4], [5], [6] called "Combination of multiple experts" (CME) has emerged to solve this problem. It is based on the idea that classifiers with different methodologies or different features can complement each other. Hence if different classifiers cooperate with each other, group decisions may reduce errors drastically and achieve a higher performance.

In general, based on output information, classifiers can be derived into two types: type-1 outputs a unique class label indicating that this class has the highest probability to which the input pattern belongs; and type-2 assigns each class label a measurement value which indicates the degree that the corresponding class pertains to the input pattern. In fact, type-2 classifiers can be transformed into type-1 ones by outputting only the class with the highest degree. This is an **information reduction** or **abstraction** process. In this sense, all classifiers are type-1 classifiers. Therefore, the research on methods of combining type-1 classifiers becomes most important.

Previous studies have developed many CME approaches of type-1 classifiers, among which the voting [7], Bayesian [5], [8], and Dempster-Shafer (D-S) [5], [9] approaches are the most representative. Simply speaking, voting is a democracy-behavior approach based on "the opinion of the majority wins". It treats classifiers equally without considering their differences in performance. The Bayesian approach uses the Bayesian formula to integrate classifiers' decisions; usually, it requires an independence assumption in order to tackle the computation of the joint probability. The D-S formula, which has frequently been applied to deal with uncertainty management and incomplete reasoning, can aggregate committed, uncommitted and ignorant beliefs. It allows one to attribute belief to subsets, as well as to individual elements of the hypothesis set. Both Bayesian and D-S approaches make use of probability to describe the different qualities of classifiers' decisions. However, in the Bayesian approach, the sum of $P(C)$ and $P(\sim C)$ is equal to one; this is not nec-

Manuscript received October 13, 1993; revised April 25, 1994. Recommended for acceptance by Dr. R. L. Kashyap.

The authors are affiliated with the Centre for Pattern Recognition and Machine Intelligence, Concordia University, Suite GM-606, 1455 de Maisonneuve Blvd. West, Montréal, Québec, H3G 1M8, Canada.

IEEE Log Number P95010.