Approximate Matching of Hand-Drawn Pictograms*

Daniel P. Lopresti
dpl@mitl.com

Andrew Tomkins
andrewt@mitl.com

Matsushita Information Technology Laboratory
182 Nassau Street
Princeton, NJ 08542-7072
USA

Abstract

We describe a new naming paradigm for pen-based computing which we call *Pictographic Naming*. Using our approach, traditional character-by-character handwriting recognition (HWX) is avoided. Instead, we use a combination of user interface conventions and approximate matching techniques. Since pictographic names incorporate pen-stroke data, they can never be reproduced exactly, so name lookup becomes a small-dictionary recognition problem. We give efficient algorithms for this problem, and show that we can accurately look up handwritten names in English and Japanese over a reasonably sized dictionary.

1 Introduction

Pen-based computers are becoming more and more prevalent. Many users find that interacting with a computer using a pen is more familiar and less intimidating than using a keyboard.

We believe that replacing a keyboard by a pen should cause a basic change in the philosophy of naming. Traditionally, a name is a short string of alphanumeric characters with the property that it can be easily stored, recognized and remembered. However, the current approach to specifying names using a pen has received widespread criticism: the user writes the name letter-by-letter into a comb or grid and the computer tries to perform HWX on each character. The HWX error rates are high enough that the user must often pause to redraw an incorrectly recognized letter. Other options seem even less appealing: the user can follow a path through a menu system until it is possible to specify a letter uniquely, or tap a pen on a simulated keyboard provided by the system. None of these methods feels like a natural way to specify a name.

Instead, we propose extending the space of acceptable names to include arbitrary hand-drawn pictures of a certain size, which we call *pictographic names*, or simply *pictograms*. The precise semantics of the pictograms are left entirely to the user. Intuitively, the major advantages of the pictogram approach to naming are the ease of specification and much larger name-space. A disadvantage is that, since people cannot be expected to recreate perfectly a previously drawn pictogram, looking up a document by name becomes an approximate ("fuzzy") matching problem. This paper studies techniques for performing this lookup.

The rest of the paper is organized as follows: first we provide more details about pictographic naming, and show how some techniques from standard operating systems, and some new techniques, can be used to speed the search for names. Next, we present Hidden Markov Models, a powerful approach to picture matching. We then show how HMM's can be used in conjunction with simpler techniques to achieve accurate algorithms for name lookup. Finally, we present data showing the effectiveness of these algorithms, and conclude.

* Presented at the *Third International Workshop on Frontiers in Handwriting Recognition*, Buffalo, NY, May 1993.

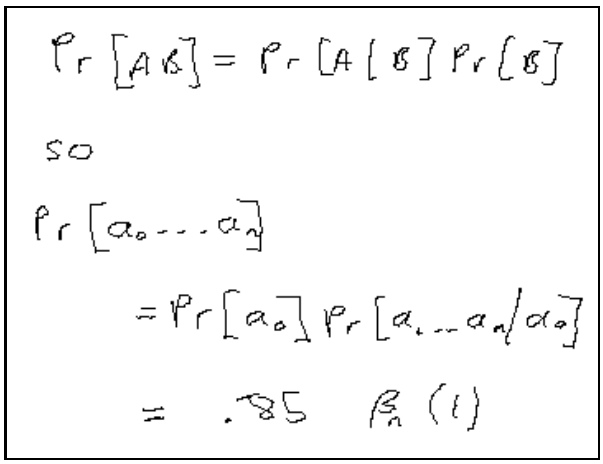


Figure 1: *An Example Document*

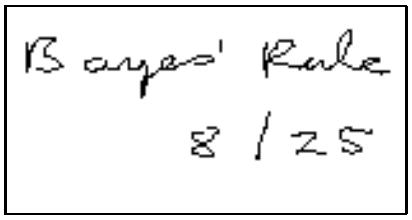


Figure 2: *Pictographic Name*

2 Pictographic Naming

2.1 Motivation

Broadly speaking, a user can specify the name of an existing file in either of two ways:

1. *Reproduction*. Duplicating the original name by retyping or redrawing it.
2. *Recognition*. Selecting the original name from a series of options in a browser, perhaps after specifying a file-path.

The motivation behind pictographic naming is the following: since the user of a GUI often specifies a file by recognizing rather than reproducing the name, HWX of the name may never be necessary and should be deferred whenever possible. A natural way to defer HWX is to leave the name as a graphical object, our pictogram, which the user can recognize later. This leaves the user free to choose complex and varied pictographic names, but forces the operating system to search for reproduced names approximately rather than exactly, a more difficult problem.

As a concrete example of pictographic naming, suppose the user has produced the diagram shown in Figure 1. Perhaps it is part of a project he/she is working on, a slide for a presentation, or something drawn to communicate an idea to a friend. The user wants to store the picture, and to be able to access it later on. A natural idea would be to write a quick, one-line note explaining the contents of the document. When the user wants to retrieve the picture, he/she can easily look through the notes and find the appropriate one, without resorting to the use of a keyboard, and without a complicated and inaccurate translation to a computer representation of letters. So, for instance, he/she might draw a quick description of the picture as shown in Figure 2. This quick description becomes the name of the document.

2.2 The Browser

We provide the user with a document browser, much like the browsers used in traditional mouse-based graphical user interfaces. However, rather than select a text string, the user selects an appropriate pictographic name. Such a browser is shown in Figure 3. Again, we should point out that pictographic names present difficulties only for the user attempting to reproduce a previously drawn name exactly, not for the user searching for a name visually.

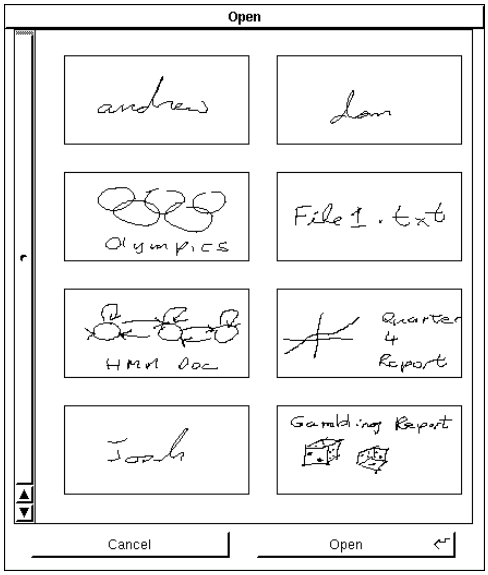


Figure 3: *Pictographic Browser*

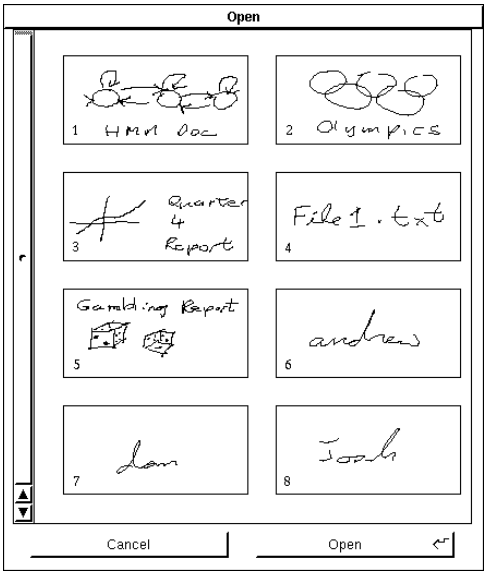


Figure 4: *Ranked Browser*

Note that the pictographic names in Figure 3 are very simple, and provide the user with far more flexibility than simple character strings. When new users are first introduced to standard file systems, they sometimes have difficulty adapting to the rigid conventions of traditional document storage and retrieval. Pictographic names allow the user to specify names rapidly and easily, while making available a much larger name space than traditional ASCII strings. Written words, sketches, non-ASCII characters, cursive script, symbols, Greek or Cyrillic letters, Kanji or other Eastern characters, or any combination of these are all valid names, as long as the user can recognize what he/she drew at a later time.

2.3 Simple Search Techniques

Several techniques from traditional operating systems have analogues in pictographically-named systems. For instance, all the typical information about a file's size, create time, update time, owner and so on will still be present, and the user can narrow the search by specifying conditions on these values.

We have implemented a graphical generalization of filename extensions (as in `report.txt`) via an approach called *Icon Tags*. At save time, the user can select various small icons from a palette; the document will be "tagged" by these icons, as shown in Figure 5. Later, the user can, for instance, narrow the search to include only documents representing correspondence by selecting the envelope tag icon. This causes the system to display only documents containing this tag.

Wildcard matching, as in `Quarter*.rpt`, can be implemented pictographically in certain cases. Say the user has written a series of letters to a friend Josh, named pictographically "Josh business", "Josh wedding", "Josh party", etc. But instead of rewriting "Josh" each time from scratch, the user copies "Josh" from the original pictographic name and writes the rest of the new name. As a result of this action, a portion of the pen stroke sequence for the related names will be identical. Unlike the approximate matching problem, which is quite difficult, exact sequence matching is relatively easy. So the user can perform the pictographic equivalent of searching for `Josh*`.

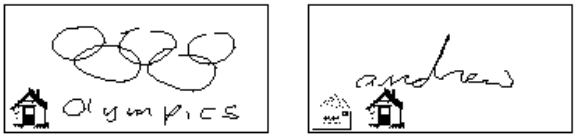


Figure 5: *Line from Browser With Tags*

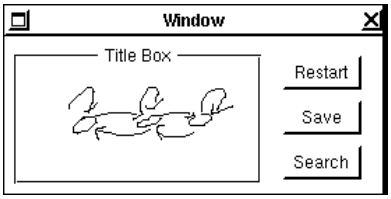


Figure 6: *Pictographic Search String*

3 Hidden Markov Models

Hidden Markov Models (HMM's) are a powerful technique for modeling uncertainty in the recognition process. For brevity, we omit a detailed description here, but an introduction to HMM's in general can be found in [4], and a more complete description of our algorithms appears in [3]. Related work using HMM's for text recognition can be found in [1] and [2].

In this section we discuss HMM's informally and show how they can be used to solve the matching problem. In Section 6 we give detailed results for HMM's and others algorithms over several databases of English and Japanese script.

Consider again the browser shown in Figure 3. Say we wish to search for the document labeled "HMM Doc". With our search routine, we can enter a search string as shown in Figure 6 where the "string" is actually a picture that need only be somewhat similar to the original filename. In particular, note that the text "HMM Doc" has been omitted from the search string. The HMM is powerful enough to recognize the remaining similarity; Figure 4 shows the resultant browser after the files have been re-ranked according to their similarity to the search string. The intended match was indeed found and ranked first. Note also the the somewhat similar looking "Olympics" filename was ranked second.

One advantage of this approach is that a search string that does not correspond point-to-point to the target name can still be matched by the warping properties of HMM's. Thus, if the query string consisted of immaculately drawn circles connected by quickly sketched arrows, an HMM would still be able to perform the match. A disadvantage of the current implementation is that the HMM attempts to match the entire search string to the entire filename, so that partially specified filenames might fail to match. Furthermore, depending on the features extracted for the HMM, there may be a level of dependence on the order in which the strokes were generated. The feature set we use in the current implementation, for example, results in a stroke-order-dependent search procedure.

3.1 Choice of Features

The current implementation extracts four features per point: direction, velocity, change of direction and change of velocity. Each feature is drawn from a set of four possible values, so the entire feature vector for a point can be represented using eight bits. This set seems to work best of the alternatives we tried, but is by no means the result of extensive testing. We are currently examining other possible feature sets to use with our matching techniques.

A carefully chosen feature set could solve several other problems. The current feature set appears to be susceptible to small variations in writing style, a dependence which could possibly be removed by a more careful choice of features. Additionally, we have been accepting input from a single manufacturer's tablet; multiple input devices with different data rates could be accomodated by a rate-independent feature set. And perhaps most importantly, a stroke-order-independent feature set would allow names to be recreated in a different stroke order, and would perhaps aid in writer independence as well.

3.2 Effectiveness

The primary difficulty with HMM's is speed. On a 68040-based NeXT machine, we require about 25 seconds to match an average-size search string against a database of 60 names. The HMM algorithm is inherently expensive, and while heuristic methods could improve performance substantially, it seems natural to investigate faster approaches that can be used in conjunction with HMM's

For a detailed discussion of the accuracy of HMM's, please refer to Section 6.

4 Discriminants

We turn now to a much simpler set of techniques, which we call *discriminants*, similar to the features of [5]. We have found that, while extremely simple to compute, discriminants alone cannot provide an effective solution to the matching problem (see Section 6). In the following section, we show how discriminants can be combined with HMM's in a hybrid scheme to retain key advantages of both.

We define a *discriminant* to be a function mapping from two images to a real number in [0..1]. The larger the number, the more similar we expect the images to be. We shall describe several discriminants, and discuss their effectiveness. For purposes of efficiency, it is convenient to consider discriminants from the following limited class: an "extraction function" is applied to each image seperately, and the two resulting values are merged via a "combiner function" to give the discriminant.

In this way, we can pre-compute the extraction function on all elements of our picture database, so to do a search we need only apply this function once to the "search pattern," then apply the inexpensive combiner function once for each element of the database.

The combiner function takes the values extracted from each of two images and combines them into a single value between 0 and 1, closer to 1 if the two values are similar. For almost all discriminants, we shall use the following simple combiner function. Say we have extracted two values $v_1, v_2 \in \mathcal{R}$. Then we define

$$\text{COMBINE}(v_1, v_2) = \frac{\min(v_1, v_2)}{\max(v_1, v_2)}$$

In the following section we shall present several discriminants and explain the extraction function that each discriminant uses. If the combiner function is not COMBINE, we give that also.

4.1 Definitions

Downstrokes: The downstroke extraction function counts the number of downstrokes in a picture. There are indications that information in English script exists primarily in the downstrokes [6], with the rest of the script essentially providing connectives. We make the trivial observation that an English word drawn many times will always tend to have a similar number of downstrokes. We define a downstroke to be a point whose direction is not *down* followed by k points whose directions are all *down*. The current implementation sets $k = 1$. The combiner function is COMBINE.

Average Direction: Each point of a figure has a direction, which is either *up*, *down*, *right* or *left*. The extraction function for the Average Direction of a figure returns a pair (x, y) such that

$$x = \Big|\{p|\text{Dir}(p) = \textit{right}\}\Big| - \Big|\{p|\text{Dir}(p) = \textit{left}\}\Big| ; y = \Big|\{p|\text{Dir}(p) = \textit{up}\}\Big| - \Big|\{p|\text{Dir}(p) = \textit{down}\}\Big|$$

Since the extraction function returns a pair rather than a single value, we cannot use COMBINE and must use a different combiner function instead.

Given two figures with average directions $\mathbf{d_1} = (x_1, y_1)$ and $\mathbf{d_2} = (x_2, y_2)$, the combiner function for Average Direction is defined to be

$$\text{Average Direction}(\mathbf{d_1},\mathbf{d_2}) = \frac{(\mathbf{d_1}^T\mathbf{d_2})^2}{|\mathbf{d_1}| \cdot |\mathbf{d_2}|}$$

Point Count: This discriminant is simply the number of points in the figure. The combiner function is COMBINE.

Curvature: The change in direction of a point, $\text{CD}(q)$, is defined as follows. Let p be the point preceeding q in the figure.

$$\text{CD}(q) = \begin{cases} 0 & \text{Dir}(p) = \text{Dir}(q) \\ 2 & \text{Dir}(p) = \text{reverse}(\text{Dir}(q)) \\ 1 & \text{otherwise} \end{cases}$$

Where *up* is the reverse of *down* and so on. We now define the curvature of a figure to be $\sum_{\text{Points } p} \text{CD}(p)$, the sum over all points of the change in direction. The combiner function is COMBINE.

Reversals: A reversal is defined to be k points in a row, no two of which are moving in opposite directions, followed by a single point moving in a new direction. The initial k points help to eliminate sampling noise, and guarantee that the stroke is moving firmly in one general direction. The final point in the reversal is the first time that the stroke has moved in the opposite direction — the first time it has reversed. The combiner function is COMBINE.

Entropy: Our current model extracts four features from each point: direction, velocity, change in direction, and change in velocity. Each of these features has an information-theoretic entropy associated with it. We can compute the entropy of each feature. For concreteness, consider direction, which is always one of {*up*,*down*,*right*,*left*}. Assume that a point's direction is independent of all previous points' directions.

Thus, we can speak of the probability of a point moving in a certain direction, for instance, Pr[*up*]. Then the *Direction Entropy* DE of a figure is defined to be

$$\text{DE} = -\sum_{\text{Directions } d} \Pr[d] \cdot \log(\Pr[d])$$

The combiner function for all entropies is COMBINE.

4.2 Effectiveness of discriminants

We have described the extraction functions used to extract a value from an image, and the combiner functions used to combine the values of two images into a single discriminant. Notice that the combiner functions all run in constant time, and typically require on the order of a single division. The extraction functions for all the non-entropy discriminants are all linear in the number of points in the image, typically requiring one or two add or compare operations per point. Images in our applications tend to contain between fifty and two hundred points each.

For these databases, a 68040-equipped NeXT machine can rank sixty items by discriminants in less than a second. Clearly, time spent computing discriminants is insignificant compared to the time required to use the HMM's of the previous section. The effectiveness of the discriminants on our data sets is described in Section 6.

5 Hybrid Schemes

We shall first consider the problem of ranking the images by using all the discriminants at once. Then we shall use this joint-discriminant procedure in conjunction with HMM's to get our final algorithm.

5.1 Combinations of Discriminants

Recall that all discriminants are in the range $[0..1]$. Say we have n discriminants total. Then a pair of images can be thought of as a vector of n values from $[0..1]$, the values of the n discriminants.

We have access to many such pairs of test data. Some will be computed from pictures that are supposed to be the same and will be labelled "+". Others will be computed from pictures that are supposed to be different and will be labelled "–". We require a procedure that can look at the vector of n values and return a single value that can be used to rank the images. It is natural to consider a weighted sum of the discriminants.

Let us represent a point in $[0..1]^n$ by $\mathbf{d} =< d_1, \ldots, d_n >$ where the d_i's are the values of the n discriminants. Then we consider ranking the images by some function f of the discriminants, defined by

$$f(\mathbf{d}) = \sum_{i=1}^{n} \alpha_i d_i$$

for some reasonable choice of the α's.

Unfortunately, computing the optimal α's using any of several reasonable definitions of "optimal" turns out to be NP-hard. We settle on a computationally efficient heuristic method based on a simple observation: it seems reasonable that "better" discriminants should have larger weight, so they contribute more to the sum. If we could define "better" we would have a simple way of choosing the α's.

So we must define a "correlation" between a discriminant and the test data. If a discriminant always has a high value whenever a point is lablelled "+", we should treat it as highly correlated. Given a search pictogram and a database, the best possible discriminant would have value 1 when applied to the search pictogram and its true match in the database, and 0 otherwise. A reasonable notion of the correlation between a given discriminant and this "perfect" discriminant would be, for instance, the sum over all test points of the squared difference between the two discriminants.

However, in our domain, a discriminant that ranks the intended figure 1,000 out of 2,000 is hardly better than one that ranks the intended figure 2,000 out of 2,000. Instead, the domain lends two natural notions of the "goodness" of a discriminant. They are the following:

1. A discriminant is good if it often selects the intended picture first for reasonable size databases, because this way the user can simply confirm the highlighted selection.
2. A discriminant is good if it often ranks the intended picture on the first page of the browser for reasonable size databases, because this way the user can easily select the picture.

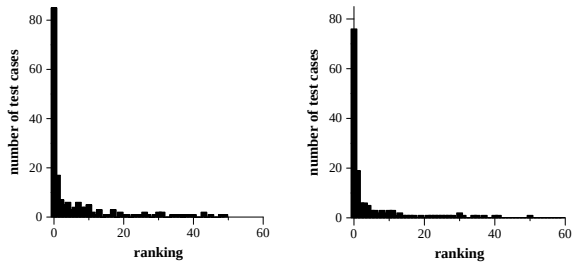
We now settle on a database-dependent notion of correlation for the purpose of comparing discriminants based on test data. We shall define the correlation of a discriminant, given a database of pictures and a set of test cases, to be a combination of the number of test cases ranked perfectly by the discriminant, and the number of test cases ranked on the first page of the browser by the discriminant. That is, given a database $\mathcal{D}$ and test set $\mathcal{T}$, the correlation of a discriminant $\mathbf{d}$ is

$$\mathit{Corr}(\mathbf{d}) = \lambda_1 \cdot \Big|\{t \in \mathcal{T} \,|\, \mathbf{d} \text{ ranks } t \text{ first}\}\Big| + \lambda_2 \cdot \Big|\{t \in \mathcal{T} \,|\, \mathbf{d} \text{ ranks } t \text{ on page 1}\}\Big|$$

In our implementation, we have chosen $\lambda_1 = 5$ and $\lambda_2 = 1$.

Then $\forall i : \alpha_i = \mathit{Corr}(\mathbf{d}_i)$

So to summarize, we compute discriminants for all pictures in the database, then rank a figure with discriminant vector $\mathbf{d}$ according to the function $f(\mathbf{d})$. The results of this procedure are shown in Section 6. The function f tends to rank the desired image amongst the top quarter of the database with high reliability. This leads us to a natural notion of a hybrid scheme.



Japanese Test Cases Ranked by HMMs English Test Cases Ranked by HMMs

Percentages of Success for HMM's				
	Database			
	1	2	3	4
Ranked On Page	72%	63%	75%	74%
Ranked First	46%	39%	47%	47%

Figure 7: *Effectiveness of HMM's*

Correlations of Discriminants			
Downstrokes	903	Average Direction	572
Point Count	1271	Curvature	904
Reversals	535	Direction Entropy	482
Velocity Entropy	348	Change in Direction Entropy	280
Change in Velocity Entropy	345		

Figure 8: *Correlations*

5.2 Combining HMM's and Discriminants

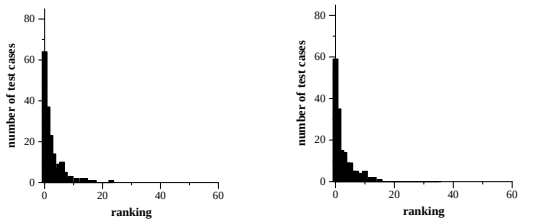
We adopt the following hybrid scheme (results are given in Section 6):

1. Compute discriminants for all elements of the database.
2. Sort all elements of the database by f.
3. Re-rank the first k (we set $k = 12$) elements of the database by HMM's.

6 Results

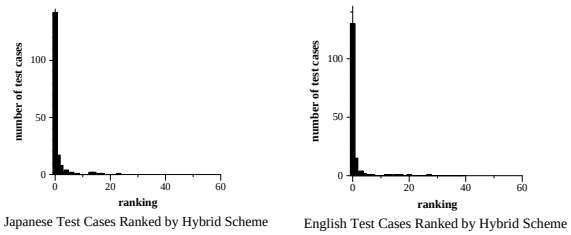
Four individuals from our lab agreed to create test databases for us. One wrote in Japanese, the others wrote in English script. Each was given a list of 60 words, and wrote each word four times. One copy of each word was used to create the browser. The other three copies were used to create a set of 180 tests to be looked up in the browser, resulting in a total of 720 browser lookups. In the tables shown below, Database 1 is Japanese, the other three are English. In the figures, we often separate the Japanese and English data – in these cases, the number of test cases in normalized. That is, all graphs assume 180 test cases total, independent of the number of databases included in the graph.

Figure 7 shows how HMM's rank the data sets for English and Japanese text. Percentages of first-place and on-page rankings are also shown in tabular format. Note that in 63% of the cases in which



Japanese Test Cases Ranked by weighted sum English Test Cases Ranked by weighted sum

Figure 9: *Effectiveness of Weighted Sums of Discriminants*



Percentages of Success for Hybrid Scheme				
	Database			
	1	2	3	4
Ranked On Page	95%	91%	94%	93%
Ranked First	77%	77%	79%	75%

Figure 10: *Effectiveness of Hybrid Scheme*

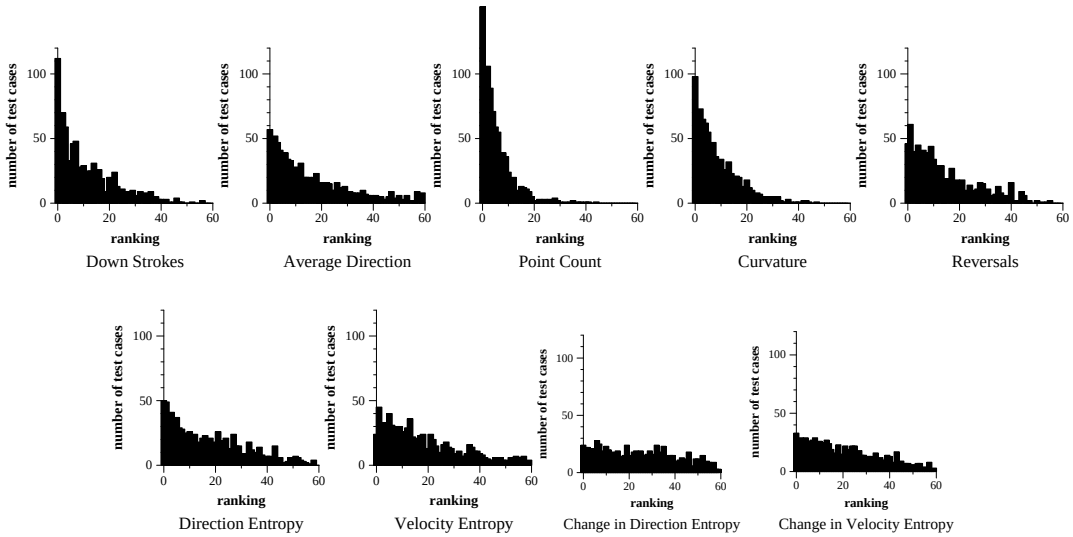


Figure 11: *Effectiveness of Individual Discriminants*

an HMM ranks the desired word within the top eight, it will in fact rank it first; this shows why our hybrid scheme, in which HMM's re-rank the top few candidates, should be effective. Figure 11 shows the effectiveness of each discriminant when used alone to rank the entire test set. All the discriminants appear to have some correlation with the true rankings. The actual correlations according to *Corr* are given in Figure 8. Down Strokes, Point Count, and Curvature seem to be particularly good measures of similarity. Note that the entropy values seem less effective; this suggests that the information-theoretic "randomness" of the strokes in a pictogram is not strongly related to the word being drawn, at least for the features we use.

Figure 9 gives the results for the weighted sum of discriminants. Note that the weighted sum of nine relatively loosely correlated discriminants turns out to be much more highly correlated than the best individual discriminant. Also, as we would expect, the weighted sum is smoother than the individual discriminants: the number of desired figures ranked first is smaller than the best discriminant alone, and the number of times the desired figure appears in the bottom two-thirds of the ranking is zero. This makes the weighted sum particularly appropriate as a first-pass in our hybrid scheme, to be followed by an accurate re-ranking determined by the HMM's.

Figure 10 shows the outcome of the hybrid scheme on English and Japanese data. Again, we present the results for English and Japanese words graphically and show the on-page and first-place rankings for each database in tabular form. As the results indicate, the on-page ranking is successful often, due to elimination of outlying points by the weighted sum procedure. Since the HMM's in the hybrid scheme are presented with a smaller domain of candidate names, the ranked-first accuracy

increases from 46% to 77%.

7 Conclusions

Pictographic Naming is a new naming paradigm for pen-based computers which defers handwriting recognition whenever possible. It has several advantages over textual naming, including the following:

- Specifying a filename using traditional character-by-character HWX is slow and error-prone.
- Pictographic names are more expressive than textual names.
- Certain languages are difficult to render using just a keyboard.
- Pictographic names naturally support language-independent systems for multilingual communities, or text-independent systems for young children or illiterate adults.

However, pictographic naming requires that the operating system be capable of performing approximate matching against a small dictionary of hand-drawn pictograms. This research has focused on finding an effective solution to this problem.

We have presented an efficient discriminant-based matching algorithm, and a more powerful but more computationally expensive HMM-based matching algorithm. We have shown how to combine these two approaches into a single scheme, achieving a tractable and accurate algorithm for fuzzy name matching. Intuitively, the discriminants provide a good "first cut" at the solution, and the HMM's refine the ranking to provide a final solution.

This approach seems quite general — we have tested the same system on English and Japanese script with similar accuracy.

References

[1] Chimnoy B. Bose and Shyh-shiaw Kuo. Connected and degraded text recognition using Hidden Markov Model. In *International Conference on Pattern Recognition*, pages B–116 – B–119, August – September 1992.

[2] Jonathan J. Hull. A Hidden Markov Model for language syntax in text recognition. In *International Conference on Pattern Recognition*, pages B–124 – B–127, August – September 1992.

[3] Daniel P. Lopresti and Andrew Tomkins. Applications of Hidden Markov Models to pen-based computing. Technical Report MITL-TR-32-92, Matsushita Information Technology Laboratory, 1992.

[4] L.R. Rabiner and B.H. Juang. An introduction to Hidden Markov Models. *IEEE ASSP Magazine January*, pages 4–16, 1986.

[5] Dean Rubine. *The Automatic Recognition of Gestures*. PhD thesis, School of Computer Science, Carnegie Mellon University, 1991.

[6] Charles C. Tappert, Ching Y. Suen, and Toru Wakahara. The state of the art in on-line handwriting recognition. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, Volume 12, No. 8:179–190, August, 1990.