# Scribble Matching

Richard Hull, Dave Reynolds, Dipankar Gupta
Office Appliance Department
HP Laboratories Bristol

HPL-94-61
July, 1994

scribble matching,
handwriting recognition,
pattern recognition

Scribble matching is a form of partial handwriting recognition that allows like scribbles to be identified. Three complimentary matching algorithms have been developed that provide in combination a matching rate of 97%. In this paper, we describe the scribble matching problem and the matching algorithms, report experimental results, and discuss applicability and future challenges.

## 1 Introduction

In the discussion of pen-based information systems, there can sometimes be a false dichotomy between leaving electronic ink uninterpreted and full handwriting recognition. In fact, there is scope for a range of partial recognition techniques that provide useful functionality without producing ASCII text. In this paper, we describe one such technique, scribble matching, that allows electronic ink to be searched for like scribbles. We show how matching rates of 97% can be achieved on unconstrained written input, enabling a variety of robust search and retrieval applications.

First, we describe the scribble matching notion and its applications, Next, we describe three complementary algorithms and their combination, and present various experimental results. Finally, we discuss the advantages and limitations of this approach, identify future work, and draw some conclusions.

## 2 What is scribble matching?

The scribble matching problem may be easily understood by considering figure 1 below. We wish to know whether two pieces of electronic ink ("scribbles") are the same in the mind of the person who wrote them. In this example, do both pieces of electronic ink represent the word "scribble", even though they are not exactly alike?

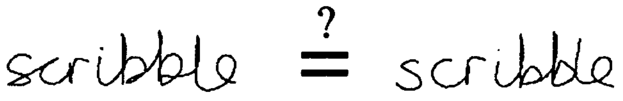


*Figure 1 - The scribble matching problem illustrated*

One way to answer this question would be to translate both scribbles into ASCII text via handwriting recognition and to compare the resulting strings. However, the reliability of such an approach would be restricted by the limited accuracy of handwriting recognition technology for unconstrained input [1]. In our approach, we circumvent the problems of full recognition by matching scribbles according to various topological and temporal features, and achieve very high matching rates. In this, we adopt an approach similar to [2, 3, 4].

The fact that scribble matching does not produce a textual version of the input may seem, at first, to limit its applicability. Certainly, it is not an appropriate technology for data entry. However, scribble matching is of great value in applications requiring search and retrieval of electronic ink. For example, scribble matching could allow users of a Personal Digital Assistant (PDA) to address messages by simply scribbling the recipient's name in an active "To" field on the message (see figure 2 below). The PDA would capture the electronic scribble and match it against the contents of a handwritten address book entered earlier. The associated telephone number of the best matching entry would be retrieved and used to send the message..

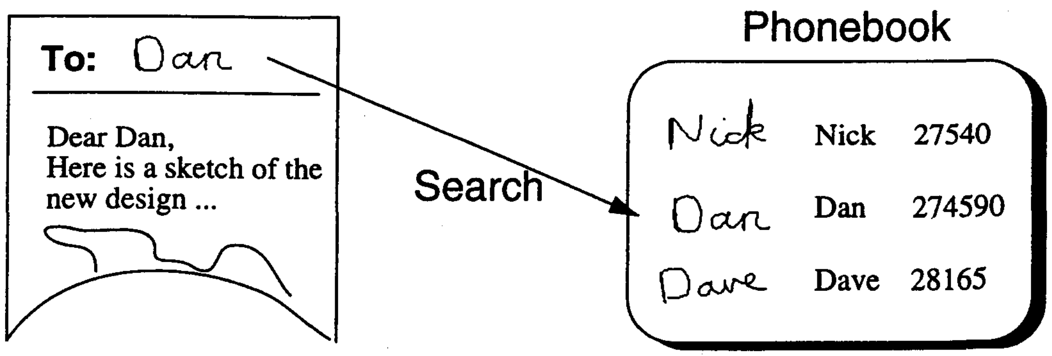


*Figure 2 - Example application for Scribble Matching*

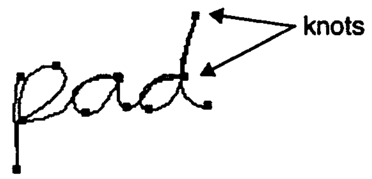
Other applications include the retrieval of documents by means of handwritten keywords or other annotations attached to the documents, and the retrieval of personal handwritten notes by matching on their contents.

## 3 Scribble matching algorithms

The overall strategy that we have adopted for scribble matching has been to identify particular features that are robust to anticipated variability, to develop and tune focussed matchers based on those features, and to combine those matchers at the system level. This has proven to be more succesful in our experience than developing a single matching algorithm based on multiple features. In this section we describe the three complementary algorithms for scribble matching that have been most effective within this strategy, and their combination.

The three matchers are termed *syntactic*, *word shape*, and *elastic* respectively, although to an extent these names reflect motivations as much as final implementations. Each scribble matching algorithm has two components, a *coder* and a *matcher*, that operate on a *scribble table* containing searchable *codes* representing, for example, the handwritten annotations on a set of documents.

The coder is used every time a new scribble is added to the table to extract a set of features from the scribble and produce an appropriate code. Each of the coding schemes is based on a set of *knot points* identified in the input scribble. These knots indicate significant points on the ink sequence - end points, cusps, ligatures and so forth - as shown below.



Each coder generates labels for the knots or for the stroke segments between knots. Clearly, the success of the scribble matching algorithms depends on the stability of knot points, and we have adopted pen velocity minima as the most stable basis for knot point detection [5, 6, 7]. Other pre-processing steps include de-bunching, de-hooking and (optionally) polygonal re-sampling.

The matcher is used when a search is to be performed. It compares the code of a handwritten *query* scribble against the codes in the scribble table. The code of the query will only rarely be an exact match to one of the entries in the table owing to the variability of the user's writing. Thus the matcher has to use some form of approximate matching to measure how similar the query code is to each table entry. The final result is a set of distance scores for each table entry which can be used to extract an ordered list of possible matches. This process is illustrated in figure 3 below.

Each of the matchers uses the same approach to approximate matching - that of string edit distances computed via dynamic programming [8, 9]. The implementation of the underlying dynamic programming engine is standard and utilises a limited beam width set by the invoking matcher. Finally, in each of the scribble matchers the matching distance is normalised by dividing it by the average of the lengths of the two strings

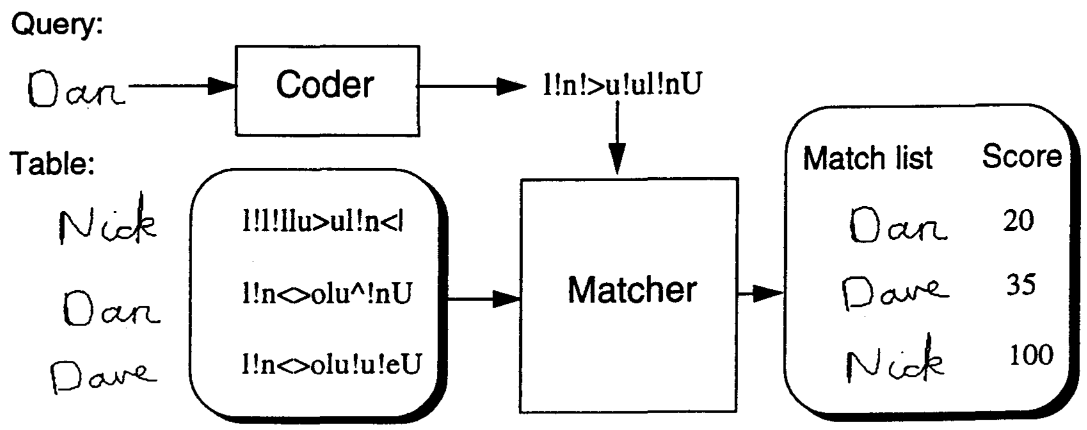
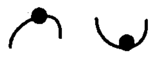
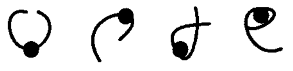


*Figure 3 - Performing a query search*

being matched. This "edit cost per knot" measure is less biased towards short match strings and in practice provides a small but useful improvement in accuracy.

## 3.1 Syntactic matcher

The syntactic matcher encodes scribbles by labelling each of the knot points with a symbol describing the local shape of the ink at that point. The labels used are:

- Cusps: **l ! < >** for up, down, left and right facing cusps.
- Open curves: **n u** for upwardly convex/concave curves. 
- Closed curves: **o p y e** 
- Miscilaneous smooth curve: **~**
- Diacritical marks: **.**
- Line end points: **l L { } N U Y**

For example, the ink would generate a labelling of: "Llp<>olu>olU".

The code is stored as a simple string with one character label for each knot point.

Matching is based on string edit distance as described earlier. The set of insertion, deletion and substitution costs are set to cope with both variability in writing style (e.g. breaks in ligatures) and instability in the labelling algorithm (e.g. the weakness of the loop tests which might label 'u' as 'o' and vice versa). The procedure for setting the edit operation costs is as follows:

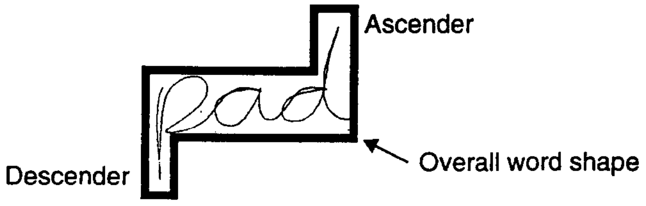
1. *Manually define a initial set of costs.*
2. *Use this set of costs to find the minimum cost edit sequence between each pair in an extensive test database of word pairs.*
3. *Measure the relative frequency $p_i$ with which each symbol or symbol pair $i$ is substituted, inserted and deleted.*
4. *Update the costs using the formula: $c_i = -\log p_i$ for each symbol or symbol pair $i$.*
5. *Test the updated costs for performance on a test set. If performance has improved further, go to step 2 else exit with prior cost set.*

*Figure 4 - Algorithm for setting string edit costs for syntactic matcher*

The result is a table of fixed costs that reflect the probability of local variability, with likely deformations such as that from a loop to a cusp incurring low costs. In practice, our first manual setting of the costs was sufficiently accurate that only one cycle of iterative improvement was needed to generate the final cost table. These costs were found to generalise well across test data sets.

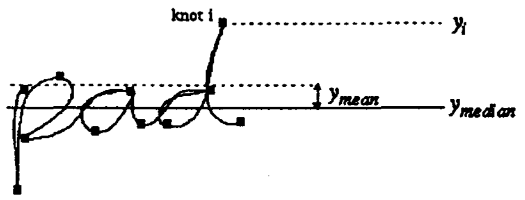
## 3.2 Word shape matcher

The word shape matcher uses a representation of the overall shapes of scribbles, reflecting the pattern of ascenders and descenders in their outer contours, eg



The coder generates a code vector of integers representing the normalized heights of knots taken in time order. Knot heights are normalized relative to a (perhaps sloping) median line, and to the mean knot height for that scribble.

More formally, a wordshape code is defined as a sequence $S$ of $N$ integers $S_i$ in time order,such that $S = S_1, S_2, \ldots, S_N$ where $S_i = \left\lfloor \frac{y_i - y_{median}}{y_{mean}} \right\rfloor$ and $y_i$ is the value of the y-coordinate of knot i, $y_{median}$ is the y-coordinate of the median line of the scribble at the same x-coordinate as knot i, and $y_{mean} = \left\lfloor \frac{1}{N} \sum_{i=1}^{N} |y_i - y_{median}| \right\rfloor$ is the mean vertical distance of knots from the median, eg



The matcher again uses the string edit distance metric. This time the cost parameters are defined by simple equations rather than by a fixed table of costs, ie

$$ins(S_i) = |S_i|$$
$$del(S_i) = |S_i|$$
$$subs(S_i, S_j) = |S_i - S_j|$$

The lowest cost edit sequences will tend to be those containing mostly substitutions of knots with similar heights. The substitution of a knot that lies at the extreme of an ascender (or descender) with any other kind of knot, or its deletion or insertion, will incur a high marginal cost and tend to increase overall matching distance. Hence, the lowest overall matching distances will indeed tend to be those between scribbles with similar patterns of ascenders, descenders and body characters, as intended. Moreover, the insertion and deletion of knots close to the median line will incur low marginal cost and will therefore not greatly affect overall matching distance. Hence, this matcher will be fairly robust to instability in the placement of knots within body characters, and in the use of ligatures, further reflecting the emphasis on the overall shape of a scribble rather than its internal details.

## 3.3 Elastic matcher

The elastic shape matcher matches the shape of the entire scribble using an approach similar to that in [10]. The main challenges to elastic matching are the sheer number of raw data points in a scribble, and the natural variability in the order of diacritical strokes such as "t-bars", and in the number and position of penlifts. The first two problems are resolved in the coder by removing diacritical strokes, and then polygonally re-sampling the remaining segments of the scribble (between knots) to reduce the number of data points. For each of the points in the subsampled space, we record three items:

- Relative height $y$ from the base of the scribble's bounding box.
- The angle $\theta$ of the tangent to the curve at the point[1].
- A classification of the point as a penlift, knot, or intermediate point.

Given this information, the matcher computes the distance between two scribbles as a linear weighted sum of the differences in the positions and tangents to the curve of each sample point in the scribbles. Note that such differences can be caused by the different locations of corresponding points on the two scribbles, or by extra points in either of the scribbles. We again use the string edit distance metric, with appropriate cost functions chosen to reflect the more general form of elastic matching[2]. The cost functions are designed to reflect the following criteria:

- Within-class[3] substitution cost is determined only by the differerences in position and angle to the curve of corresponding points, ie for points *p, q* :

  $subs(p,q) = m(y_p - y_q) + n(\theta_p - \theta_q)$ where *m* and *n* are appropriate weights.
- The cost of substituting an intermediate point for a knot, or a knot for a penlift is further increased by some weighting factor to reflect the greater deformation implied, but bounded to reflect errors in knot finding and natural variability in penlifts.
- The cost of substituting an intermediate point for a penlift is made yet higher to reflect the most severe form of deformation.

In effect, the higher cost of substituting an intermediate point for a knot (or penlift) encourages the matching algorithm to try to line up knots in the two scribbles, even if it means requiring a number of insertions or deletions of intermediate points. Insertion and deletion costs are symmetrical and combine to give a greater cost than a corresponding substitution. The insertion/deletion cost for a penlift is set highest and made equal to the cost of substituting the penlift for an intermediate point.

## 3.4 Combining scribble matchers

The three scribble matchers described may be combined in many ways, for example in a pipeline, or in parallel. However, in this paper, we shall be concerned only with one very straightforward combination based on a linear sum of matching distances. See [11] for a more far-ranging and detailed account of multiple matcher architectures.

This combination technique simply uses each matcher to find a distance from the query scribble to every entry in the scribble table, normalizes the distances to a common scale[4], and sums them to produce an overall distance *d*, ie

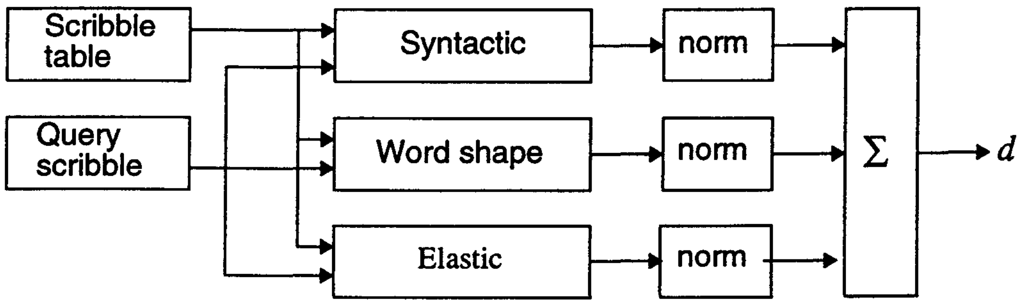


*Figure 5 - Overview of combination scheme*

The effect of normalization is to place all match distances on a single scale that may be interpreted as the degree of confidence of match. Thus the overall normalized distance could be used to reject low confidence

1. If this is a penlift, we compute the angle between the last point of the previous stroke and the first point of the next.
2. In which repeated insertions of a sequence of identical symbols incur no (or little) extra cost.
3. ie knot for knot, point for point, or penlift for penlift.
4. by dividing by the standard deviation of that matcher's best match distance

matches. In practice, the gap between the best and second-best match was found to provide a better rejection heuristic.

## 4 Experimental procedure

The scribble matching algorithms have been tested on data collected from around thirty subjects. Subjects were instructed to write as naturally as possible, and the resulting corpus contains cursive, printed and mixed styles. The collection procedure was designed to allow a study of the stability of an individual's handwriting over time and captured multiple instances of the same scribbles over five months. The subjects were allocated to development and test sets and used accordingly.

### 4.1 Test data

#### Data set

The data set consists of 100 items comprising one, two or three names and/or sets of initials, such as *Andrew, Elizabeth Taylor, W.A.Mozart,* and *IBM.* The data set was designed to give minimal alphabet coverage in that it included at least one instance of every letter in upper and lower case, but it was not balanced to reflect language statistics.

#### Subjects

In order to collect multiple instances of the data set, each subject was asked to write the specified data set twice in each of three sessions. The sessions were spaced one week and twenty-one weeks apart. Not all subjects attended every session, but the outcome was a test corpus containing 6 instances of the dataset from 19 writers, 4 instances from a further 12 writers, and 2 instances from one further writer. The subjects, drawn from students and staff at a local university, were roughly balanced in terms of gender. All but a few were right-handed, and most were educated in Great Britain.

The data collected from each subject was assigned to one of three distinct sets, one used for development of the matching algorithms[1] , and two test sets T2[2] and T3[3].

#### Ink characteristics

The data was collected on a Wacom PL-100V active tablet connected to a 25MHz 386 PC running Windows-for-Pen. For various non-technical reasons, the data was collected using a mouse driver rather than a pen driver. This results in lower quality data in terms both of spatial and temporal resolution. In particular, ink is collected at VGA resolution which is about 1/6th of digitizer resolution, and appears bunched in time rather than being regularly clocked. From a methodological point of view this results in a harder test set than if it had been collected via a pen driver and will tend to result in an underestimate of matching performance rather than an overestimate.

### 4.2 Performance metrics

The principal measurement made over the data sets was of scribble matching performance. This is measured as follows. For a particular writer, two scribbled instances of each item in the data set are selected[4] and loaded into a scribble table. Hence, after loading, the scribble table has 200 entries. Each of the loaded scribbles in turn is then taken temporarily out of the table, designated the *query*, and matched against the 199 scribbles remaining in the table. The table is sorted according to each scribble's matching distance from the query, in ascending order. The scribble at the head of the sorted table is designated the *best match.* The scribble that is the second instance of the data set item represented by the query is designated the *intended match.* If the intended match is the best match, then the *Top1* count is incremented. If the intended match is within the best five matches, the *Top5* count is incremented. The procedure is repeated for all writers and statistics are produced.

1. A set of 11 writers selected randomly from those attending at least the first session
2. A further 8 writers who attended the first 2 collection sessions.
3. A further 12 writers who attended all 3 sessions.
4. For example, from sessions a week apart.

## 5 Results

In this section, we report on a number of experimental results obtained for the scribble matching algorithms according to the procedure described in the previous section.

### Single matchers v combined matcher

Table 1 shows the overall Top1 and Top5 performance of the various matchers on the development writers,

**Table 1: Overall perfomance of scribble matchers**

| Data set | Syntactic | | Word Shape | | Elastic | | Combination | |
|---|---|---|---|---|---|---|---|---|
| | Top1 | Top5 | Top1 | Top5 | Top1 | Top5 | Top1 | Top5 |
| Development (11) | 91.7 | 97.8 | 89.4 | 96.1 | 97.3 | 99.5 | 98.1 | 99.8 |
| Test set T2 (8) | 86.9 | 95.6 | 88.6 | 96.4 | 95.8 | 98.9 | 97.3 | 99.5 |
| Test set T3 (12) | 87.8 | 96.2 | 87.2 | 95.4 | 95.1 | 98.4 | 96.9 | 99.1 |

and on the two sets of test writers. These figures relate to matching over pairs of scribbles captured during the second session, giving a scribble table of 200 entries. We may make several observations

- The elastic matcher performs better than the syntactic and wordshape matchers, though at the cost of much greater computation [11]. As expected, the combined matcher performs best of all.
- For the combined matcher, results on the test sets are close to those on the development set. The algorithms do not seem to be over-trained on the development set and should generalize to a wider population.
- The Top5 perfomance of the combined matcher means that the correct hit can virtually always be found in a short list of near matches that may be presented when a user signals a bad best match.

The beneficial effect of combining the individual matchers is confirmed by the graph in figure 6. This shows that combining matchers not only increases the average performance over a set of writers but also greatly reduces the spread of performance experienced by different users.

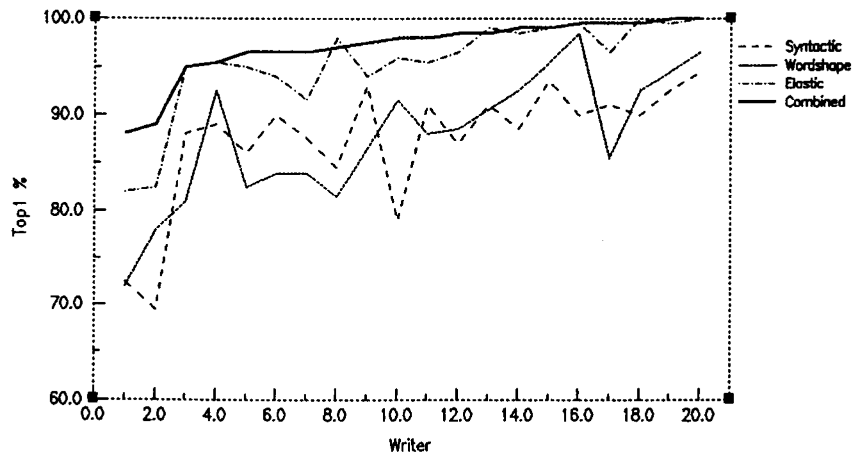


*Figure 6 - Per-writer perfomance of scribble matchers on test writers*

## Stability over time

One possible limitation of scribble matching could arise from long-term drift in an individual's handwriting. However, Table 2 shows that, in fact, long-term degradation of matching performance for the available test writers is acceptably low.

**Table 2: Variation over time Top1 - Top5**

| Test set T3 | Syntactic | | Word Shape | | Elastic | | Combination | |
|---|---|---|---|---|---|---|---|---|
| | Top1 | Top5 | Top1 | Top5 | Top1 | Top5 | Top1 | Top5 |
| Same day[a] | 87.8 | 96.1 | 87.2 | 95.4 | 95.1 | 98.4 | 96.9 | 99.1 |
| 1 week | 85.9 | 95.7 | 86.2 | 95.8 | 94.2 | 98.5 | 97.2 | 99.4 |
| 5 months | 82.3 | 94.6 | 84.4 | 94.9 | 95.3 | 98.3 | 96.8 | 99.2 |

a. Again, this result is reported for the second session. The first session gives poorer correlation with the 5 month session, possibly due to subjects' lack of familiarity with the system.

As may be seen, there is some reduction in performance for the syntactic and wordshape algorithms when matching scribbles collected five months apart. However, the elastic matcher is very stable, and the combined algorithm shows no degradation as the interval between scribble capture increases. These results suggest that there may be some underlying change in the writing styles of individuals over the five month period, but that the combined scribble matcher is robust enough to cope. Note that if long-term drift is revealed to be a problem, then a systems-level solution may be most appropriate, for example replacing the reference scribble in the scribble table with the query scribble after a successful match

## 6 Discussion

For many applications, scribble matching has several advantages over an approach based on full recognition -

- For a personal device (so that the search is comparing one person's writing with their own writing) then the search is more accurate. Handwriting recognition accuracy is currently limited in its ability to cope with the huge variety of letter forms people employ in natural writing. However, scribble matching simply requires the writer to use a similar letter form when writing the keyword each time. It doesn't matter if that letter form is not recognisable.
- For small search lists (a few hundred entries) of unconstrained writing, scribble matching is computationally cheaper than translation to text.
- The writer is not restricted to using only letters from (say) the Roman alphabet, arbitrary symbols can be freely used in annotations and then later used in a search without any need for training. Thus small iconic pictures, personal symbols (such as bullet marks) and symbols from other languages can be freely used in annotations so long as they are sufficiently distinct and stable.

In realizing the advantages, we have had to address two main challenges -

- An individual's writing does vary, for example in the use of ligatures, even when writing the same words. Scribble matching must be robust against such variation.
- Scribble matching is based on the assumption that a writer's style is reasonably stable. However, a person's writing does change a little over time and scribble matching techniques must be robust against this small variation.

In the results section, we have shown how these challenges have been overcome to produce an accurate and robust scribble matching system. An overall Top1 performance of 97% (Top5 >99%) can be achieved by combining complementary scribble matching algorithms. The combined scribble matcher is robust against long-term drift in handwriting, or at least that resulting from the passage of five months. Longer intervals are likely to produce greater degradation but this problem may be best tackled at the system level, for example by updating the reference scribbles held in the scribble table after successful matches.

Moreover, these results have been achieved using poor quality ink captured at VGA resolution and with partial loss of timing information. The algorithms may be implemented efficiently [11] and have been demonstrated on mid-range PCs with 386/25 processors.

There remain, however, other challenges requiring future work. The results given in this paper have been produced with scribble tables containing 200 entries. While large enough to accurately represent many applications of interest, this scale of test is insufficient for assessing performance in applications requiring search over large volumes of ink, such as might be found in a folder of electronic notes. In additon to further improving the efficiency of our implementations, one approach to large-scale search may be to pre-compute matching indices when electronic notes are first filed, so restricting real-time search to a small number of representative scribbles indexed to many instances scattered thoughout the notes.

A second focus for future work would be to extend scribble matching to offline data. This would require significant modification to the knot finding algortihm underlying each of the matchers. The current algorithm exploits dynamic information in electronic ink (the pen velocity) that is not present in the static images usually found in offline settings. One approach might be to find other significant and stable points on a static scribble, such as y-extrema. However, an alternative approach of interest might be to infer the dynamic information from the static image (eg see [12]).

## 7 Conclusions

A set of algorithms have been developed for the relatively new problem area of scribble matching. Matching rates of around 97% have been achieved over mixed cursive and printed input collected from thirty writers over a period of five months, despite the use of low-grade electronic ink. The combined matching algorithm is robust to any changes in subjects' handwriting styles that may have occurred over the collection period. Current performance is sufficient to enable many search applications of interest, such as the retrieval of electronic documents with scribbled annotations. Several future directions have been identified for this work, including extensions into large-volume search and offline matching.

## 8 References

[1] Charles C. Tappert, Ching Y. Suen and Toru Wakahara. The state of the art in on-line handwriting recognition. *IEEE Transactions on Pattern Recognition and Machine Intelligence.* 12(8):787 - 808, August, 1990.

[2] Daniel P. Lopresti and Andrew Tomkins. *Pictographic naming.* Technical Report 007-mei-pti-mitl-228-1, Matsushita Information Technology Laboratory, Princeton, November, 1992.

[3] Thierry Paquet and Yves Lecourtier. Recognition of handwritten sentences using a restricted lexicon. *Pattern Recognition.* 26(3):391 - 407, 1993.

[4] Daniel P. Lopresti and Andrew Tomkins. *A comparison of techniques for graphical database queries.* Technical Report MITL-TR-45-93, Matsushita Information Technology Laboratory, Princeton, May, 1993.

[5] Hans-Leo Teulings, Lambert Schomaker, Pietro Morasso and Arnold Thomassen. Handwriting-analysis system. *Proceedings 3rd International Symposium on Handwriting Computer Applications*, pages 181 - 183. July, 1987.

[6] Lambert Schomaker. *Simulation and recognition of handwriting movements.* PhD Thesis, Nijmegen Institute for Cognition Research and Information Technology, 1991.

[7] Hans-Leo Teulings, Lambert Schomaker, Gerben Abbink and Eric Helsper. Invariant segmentation of on-line cursive script. *Proceedings Sixth International Conference on Handwriting and Drawing, ICOHD'93, Paris*, pages 198 - 200. 1993.

[8] David Sankoff and Joseph B. Kruskal (editor). *Time warps, string edits, and macromolecules: The theory and practice of sequence comparison.* Addison-Wesley, 1983.

[9] Okuda et al. Correction of garbled words based on Levenstein metric. *IEEE transactions on computing.* C-25(2):??, Feb, 1976.

[10] C.C. Tappert. Speed, accuracy and flexibility trade-offs in on-line character recognition. *International Journal of Pattern Recognition and Artificial Intelligence.* 5(1-2):79 - 96, June, 1991.

[11] D.E.Reynolds, D. Gupta and R. Hull. Architectures for efficient scribble matching. *Submitted to 4th Int. Wkshp on Frontiers of Handwriting Recognition*, 1994.

[12] David S. Doermann and Azriel Rozenfeld. Recovery of temporal information from static images of handwriting. *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition* , pages 162-168. 1992.