Efficient Index for Handwritten Text

Ibrahim Kamel

Dept. of Electrical and Computer Engineering
University of Sharjah
kamel@sharjah.ac.ae

Abstract. This paper deals with one of the new emerging multimedia data types, namely, handwritten cursive text. The paper presents two indexing methods for searching a collection of cursive handwriting. The first index, word-level index, treats word as pictogram and uses global features for retrieval. The word-level index is suitable for large collection of cursive text. While the second one, called stroke-level index, treats the word as a set of strokes. The stroke-level index is more accurate, but more costly than the word level index. Each word (or stroke) can be described with a set of features and, thus, can be stored as points in the feature space. The Karhunen-Loeve transform is then used to minimize the number of features used (data dimensionality) and thus the index size. Feature vectors are stored in an R-tree. We implemented both indexes and carried many simulation experiments to measure the effectiveness and the cost of the search algorithm. The proposed indexes achieve substantial saving in the search time over the sequential search. Moreover, the proposed indexes improve the matching rate up to 46% over the sequential search.

1 Introduction

The handling of handwritten text has recently increased in importance, especially with the widespread of personal digital assistants (PDA) and tablet computers. This allows the user to store data in the form of handwritten notes and formulate queries based on handwritten samples. For example, searching a large database which contains one or more handwritten fields (e.g., verifying signature of the bank accounts.)

One way to handle the handwritten text is to translate it first into ASCII-equivalent characters using pattern recognition techniques and then to store it as ASCII text. Similarly, the search algorithm translates the query string into a sequence of ASCII characters and then performs a traditional search through the database. Thus, the recognition phase is an intermediate step between the input device (pen and tablet) and the storage device. But this is not practical because of the latency delay introduced by the recognition step. Moreover, accuracy of the recognition of cursive writing is low and thus will result in high error rate. It is difficult even to identify letter boundaries in the cursive string. Moreover, by translating the handwritten string into a sequence of predefined symbols (alphabet), we lose much information, such as the particular shape of the letter "allograph", the writing style, etc. Another disadvantage to this method is that the recognition phase renders the system sensitive to the underlying language.

D. Ślęzak et al. (Eds.): MulGrab 2009, CCIS 60, pp. 25–36, 2009.

A more natural way to handle the handwritten text is to treat it as a first-class data type. The handwritten string is treated as a pictographic pattern without an attempt to understand it. During the search process, the query string is compared to database strings using an appropriate distance function. This gives the user more expressive power; he can use non-ASCII symbols, drawings, equations, other languages, etc. Searching in handwritten cursive text is a challenging problem. A word written by two different people cannot look exactly the same. Moreover, a person cannot recreate perfectly even his own previously drawn word. Hence, exact match query will not be appropriate and similarity (or approximate string matching) would be more suitable in this case. The search algorithm should look for all strings which are "similar" to the query string. One additional requirement for pen-based and/or personal digital assistant environment is the need to support online retrieval and fast response time.

In this paper we address the problem of searching for a given cursive string in a database of handwritten text. The rest of the paper is organized as follow. Section 2 describes the proposed word-level index while section 3 presents the proposed stroke-level index. Prior works related to cursive handwriting is briefly described in section 4. In Section 5 we show experiments for measuring the response time and the matching rate of the proposed indexes. Section 6 gives our conclusions and future work.

2 The Proposed Word-Level Index

In this section we propose a two-step indexing schema to index a large repository of handwritten cursive text. The proposed index treats each word as a pictogram (or an image) and it consists of two steps, the filtering step and the refinement step. In the first step we use a coarse index that filters out most of the unwanted pictograms and produces a set of pictograms called the *candidate* set. The second step uses a sequential algorithm that operates on the candidate set to find the best k matches to the query word which are reported as the final answer.

The filtering step uses global features to characterize the different pictograms (words). A set of features $f_1, f_2, \ldots, f_x$ (described in Section 1.1) are calculated for each pictogram in the database as well as for the queries. Thus, each pictogram can be represented as a multidimensional point in x-dimensional space. A good set of features maps two instances of the same pictogram to points that are close in the multidimensional space. At the same time, two different pictograms should be positioned as far apart as possible.

These points are organized in a multidimensional index. We choose the R-tree because of its ability to prune the search space at early levels of the tree structure and because of the guarantee of good space utilization.

Edit distance: is a distance function that quantifies the similarity between two text strings. The edit distance used in [LT94] aligns the two strings and transforms one string into the other using the following operations:

- deletion of a symbol.
- insertion of a symbol.
- substitution of a symbol by another one.
- splitting of a symbol into two.
- merging two symbols into one.

Each of these operations has a predefined cost associated with it. The weighted cost of the transformation is used as a distance between the two strings. We call this distance metric ED.

Inflection distance: we define the inflection distance in the same way the edit distance is defined. Ink can also be represented by a sequence of codewords that represent the inflection points of the pictogram. An inflection point marks the change of direction in the pictogram, e.g., going from an upwards direction to a downwards direction, or from a leftward direction to a rightward one. We define eight inflection symbols. Using the inflection representation, we can also compute an edit distance between two pictograms. For this, we assume that only insertions and deletions are allowed and that each operation (insertion or deletion) has an associated cost of 1. We call this *inflection distance* (ID).

To find the pictograms that are similar to a given query pictogram, we map the query to a point in the x-dimensional space, by extracting x global features. Then, we perform a similarity search. The candidate set is formed by the multidimensional points (i.e., pictograms) returned by the query. In the refinement step, we apply a sequential algorithm to the candidate set to find the best k matches to the query word. We use a combination of the two metrics ID and ED. Our experiments show that the cost of the ID comparison is about 5 times less than the cost of ED comparison. Thus, we use the ID as a second stage filter in the retrieval process. The procedure is as follows. First, we perform pair wise comparison (sequential search) of the query against each of the retrieved pictograms by using ID. This results in a ranked set S of pictograms. Then, we take the best m (in the experiments we set m = 20%) elements of S and use the metric ED+ID to perform another round of sequential comparison. We call this procedure EID. Increasing m improves the retrieval rate but it increases the total response time. The value of m can be determined experimentally. After that, the best k matches are presented to the user.

Two instances of the same word will have two different values v_i, $\acute{v}_i$ for feature f_i. To accommodate for this variability, the length l_i of the query hyper-rectangle along feature (axis) f_i should satisfy:

$$l_i \geq v_i - \acute{v}_i$$

2.1 Global Features

The eight global features are used in our indexing scheme are: number of strokes, number of points, number of vertical inversions, total-change-MBR-height, avg-weighted-MBR-area, number of thin strokes, X-centroid, Y–centroid.

This word-level index is suitable for searching large number of cursive handwriting. In Section 5, we perform simulation experiments to measure the effectiveness of the proposed index in filtering unwanted pictograms and identifying the most similar word.

3 The Proposed Stroke-Based Index

In this section, we propose an index that is suitable for small database of cursive handwriting. Unlike the word-level index, the index compares text at the stroke level

rather than word level. Thus, this index is more costly than the word level index but it is more accurate in identifying the query work. The stroke-based index allows fast retrieval of similar strings and can handle insertion, deletion, m-n substitution errors and substring matching. This index is dynamic in the sense that insertion and deletion operations can be intermixed in real time with the search operations. Given a search query string, the answer would be a set of the strings or substrings that look like the query string.

3.1 The Basic Idea

We model the cursive string as a sequence of strokes. Each stroke is described by a set of features and thus can be represented by a point in the multidimensional feature space. We propose to store these points in a multi-dimensional index, and more specifically, the R-tree because of its ability to prune the search space at early levels of the tree structure and because of the guarantee of good space utilization. A cursive string is read from the tablet as a sequence of points in real time. Each point is represented by the tuple (x, y, t) where x, y are the coordinates of the point in two-dimensional space and t is the time at which the point is printed. These points are grouped into strokes. A stroke ends and a new one starts at each local minimum in the x-y coordinates. This method is known as local minima segmentation.

We describe each stroke with a set of 11 features. The features, which have been described in [15], describe the geometric properties of the stroke, e.g., the length of the stroke, the total angle traversed, and the angle and length of the bounding box diagonal. The features are selected so that strokes that look alike tend to have similar vector values according to some distance functions. Due to the variability in handwriting, the feature vectors that correspond to different instances of one stroke tend to vary slightly. Vectors that represent different instances of the same stroke form a cluster in the feature space. Thus, strokes that look similar will have their representative clusters close to each other or even overlapping in the multi-dimensional space.

Given a string S, the stroke segmentation program decomposes S into a sequence of t stokes S_i, $1 > i < t$. Each stroke S_i is represented as a point in an 11-dimensional space formed by the features f_1, $f2$, ... , f_{11}. The string S is represented by t points in the space. These multi-dimensional points (=strokes) are stored in an R-tree index. Each R-tree node occupies one disk page. Non-leaf nodes, which are small in number, are kept in main memory, while leaf nodes are stored on the disk. A set of points that are close to each other will be stored in the same leaf node (level 0 in the tree). Each entry in the leaf node in the form of (word-id, P) contains the coordinates of a point P (=stroke) and a word-id for the pictographic description of the string that contains (owns) this stroke (see Figure 1). Non-leaf nodes in level i, where $i > 0$, have entries of the form (ptr, R) where ptr points to a child node and R is the Minimum Bounding Rectangle (MBR) that encloses all the entries in the child node.

3.2 Similarity Search Queries

To search for a string Q, we treat the query string in a manner similar to that described in the previous section. The set of strokes q_1, q_2, ... , q_x: are extracted from Q. Since it is impossible to write the same word twice identically, we need similarity

queries. For each stroke q_i, a range query in the form of a hyper-rectangle is formed in the 11-dimensional space. The center of the hyper-rectangle is the query point, and the length along each axis is a ratio (2 x p) of the standard deviation of the data along that axis.

The output of each query would be a set of word-ids for those words which contain a stroke similar to the query stroke. We call this set the candidate set C. We then apply a simple voting algorithm as follows. Each word-id takes a score that indicates how many times it has appeared as an answer for the queries q_i, $1 < i < x$. The set of word-ids that have the highest scores are reported as the answer. Note that we did not use any expensive operations, nor did we access any of the pictographic representation of the strings from the database.

Algorithm Search (**node** Root, **string** Q):
S1. *Preprocessing*:
Use Q to build the set of strokes $q_1, q_2, \ldots, q_x$.
Extract the set of features for each q_i, $1 < i < x$.
S2. *Search the index*:
For each stroke q_i, perform a range query.
Form the candidate sets.
53. *Voting algorithm*:
Words with the highest score are the answer.

3.3 Feature Space Dimensionality

Our goal here is to reduce the number of features needed to describe the stroke by transforming the data points into another space with smaller dimensions. This problem is known as dimensionality reduction. Until now, we used 11 highly correlated features to describe each stroke. We use the *Karhunen-Loève* transform [5], also known as *Hotelling* transform or *Principal Component Analysis*) to reduce the dimensionality of the feature space. The transform maps a set of vectors to a new space with an orthogonal uncorrelated axis. The *Karhunen-Loève* transform consigns most of the discrimination power to the first few axes. Hopefully, using only k axes, k < 11, we lose little information while reducing the index size significantly.

The axes of the new feature space are the Eigen vectors of the auto correlation (covariance) matrix for the set of data points. The *Karhunen-Loève* transform sorts the eigenvectors in decreasing order according to the eigen values and approximates each data vector with its projections on the first k eigenvectors, $k < 11$.

We collect a small sample from the writer in advance and apply the *Karhunen-Loève* transform to calculate the vector transformation matrix. All strokes (vectors) are mapped to the new space and then inserted in the index.

3.4 Reducing the Candidate Set Size

Two strings are similar if they have similar strokes in the same order. The output of the search query gives a set of strings which has strokes similar to the query stroke but they do not necessarily occur in the same location. The candidate set is thus large because it contains many false candidates. Moreover, the voting algorithm does not take into consideration the location of the stroke.

To make use of the stroke location and to reduce the size of the candidate set, we store the location of the stroke inside the string as one more dimension in the feature space. Each stroke is then represented by k features f_1, $f2$, ... , f_k and by its location stk_{loc} inside the string in (k + 1) dimensional space.

Two instances of the same string will not, in general, have equal numbers of strokes. The difference, however, is expected to be small. Thus, the answer to the range query that corresponds to stroke q_i should include strings that have similar strokes not only in the position i but also in a window of length w around i. We found experimentally that $w = 3$ gives the best results (thus covering stroke numbers $i - 1$, i, and $i + 1$).

In substring matching, however, we want to allow the query string to start at any position inside the database string. In this case, a partial match query rather than a range is used. In a partial match query, the extent of the query rectangle is specified for all axes f_1, $f2$, ... , f_k as before. For the stroke location stk_{loc} axis, the extent is left open ($-\infty$, $+\infty$) to allow the query string to start at any position inside the database string. Otherwise, the algorithm is similar to that for similarity query.

4 Prior Work

There are many researches on modeling, retrieval, and annotation of cursive handwriting. However, there are not many works on indexing cursive handwritten text. Research included handling different languages like Arabic [17], Chinese [12], and Indian languages [2] and on-line handwriting [8].

[16] proposed a technique that is based on an additive fusion resulted after a novel combination of two different modes of word image normalization and robust hybrid feature extraction. They employ two types of features in a hybrid fashion. The first one, divides the word image into a set of zones and calculates the density of the character pixels in each zone. In the second type of features, they calculate the area that is formed from the projections of the upper and lower profile of the word.

[11] proposed a method for generating a large database of cursive handwriting. Synthesized data are used to enlarge the training set. He proposed method learns the shape deformation characteristics of handwriting from real samples; then used for handwriting synthesis.

[4] also proposed a method to synthesize cursive handwriting of the user's personal handwriting style, by combining shape and physical models together. In the training process, some sample paragraphs written by the user are collected and these cursive handwriting samples are segmented into individual characters by using a two-level writer-independent segmentation algorithm. Samples for each letter are then aligned and trained using shape models.

[7] word-spotting system operates on a database containing a number of handwritten pages. The method used for word matching is based on a string matching technique, called dynamic time warping. Dynamic Time Warping (DTW). The following three features are computed at each sample point in the word, resulting in a sequence of feature vectors: The height (y) of the sample point: This is the distance of the sample point from the base of the word; the stroke direction; and the curvature of the stroke at point p. The word to be compared is first scaled so that it is of the same size

(height) as the keyword, and translated so that both the words have the same centroid. The DTW technique then aligns the feature vector sequence from a database word to that of the keyword using a dynamic programming-based algorithm. The algorithm computes a distance score for matching points by finding the Euclidean distance between corresponding feature vectors and penalizes missing or spurious points in the word being tested.

5 Experimental Results

This section presents experimental results that show the effectiveness of our proposed indexes. The two proposed methods are implemented in C. Our database consists of 8,000 handwritten cursive words produced by one writer. The same writer then recreated 100 words to be used as queries. Since our data is static (no insertion nor deletion) we used the Hilbert packed R-tree [9] because of its high space utilization. For dynamic data, other R-tree variants that allow insertions and deletions can be used (such as R*tree [1], and the Hilbert R-tree [10].

5.1 Evaluation of the Global Features

In this section we evaluate how good the set of global features, listed in Section 1.1, are in pruning the search space and retrieving the most similar set of words. We store the leaf nodes (which account for the large portion of the R-tree) on the disk and keep the non-leaf nodes in main memory (non-leaf nodes occupy about 50 K-bytes for 8000 words.) Figure 1 shows the percentage of the database that is filtered out by the R-tree as a function of the database size. Notice that the pruning capability is increasing with increasing the database size. The reason for this is that the query size is constant regardless of the database size (recall that the query size is defined by the characteristics of the user handwriting.)

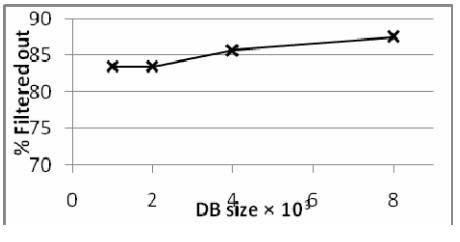

Fig. 1. The percentage of the database filtered out by the R-tree

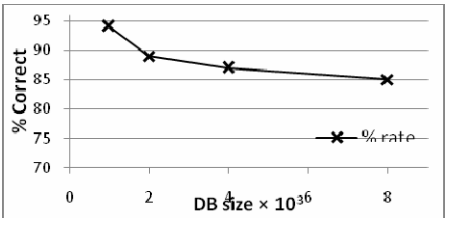

Fig. 2. The retrieval rate of the R-tree only

To see how good the global features are in describing the cursive handwriting, we show in Figure 2 the percentage of cases in which the correct answer to queries are in the candidate sets (R-tree retrieval rate). The graph show high retrieval rate 86% - 95%. As expected, the retrieval rate decreases with the size of the database.

5.2 Comparison between the Proposed Index and the Sequential Scan

In this section we compare the proposed schema (R-tree + EID) with ED and EID sequential searches.

Figure 3 shows the total search time per query for various database sizes. The above R-tree package stored the tree in main-memory; thus, we had to simulate each disk access with a 15 msec delay. For our method (marked as "R-tree+EID"), it shows the time per query after searching the R-tree and screening the resulting subset of pictograms with EID. We compared our method with the ED sequential algorithm. The figure also shows the time it takes to perform sequential search over the entire database using our EID sequential algorithm. Our method "Rtree+ EID" outperforms ED sequential search in the entire range of database sizes. For 8,000 pictograms the ratio of search times is 12:1. We included in the graph the performance of our sequential EID (no Rtree) which also outperform ED. Note that, for small databases 1000 or less, EID is little faster than "Rtree + EID". This is because of the overhead of the R-tree (the relative cost of node access increases with decreasing the database size). As expected the sequential search times grow linearly with the size of the database, while for our method "R-tree+EID" the search times grow sub-linearly in the entire range. Figure 4 and Figure 5 plot the matching rates obtained when showing the best k = 3 and k = 5 pictograms respectively. As we can see, the matching rates for the index outperform those of sequential search.

Figure 6 and Figure 7 explain the sub-linear behavior of our method. Figure 6 shows the percentage of pictograms returned by querying the R-tree. As we can see, the relative size of the subset obtained by searching the tree decreases with increasing

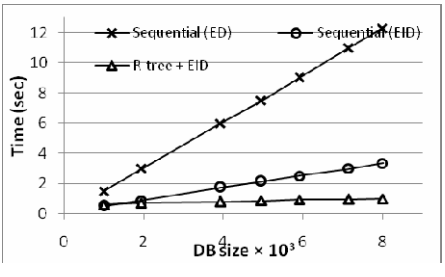

Fig. 3. Total search time

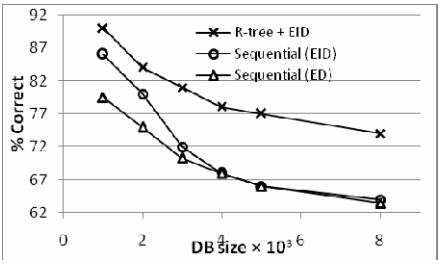

Fig. 4. Matching rate (top 5)

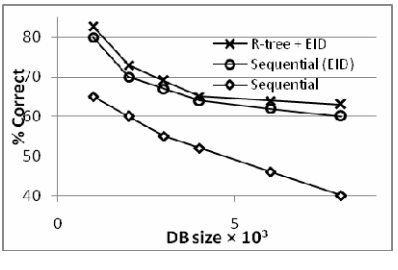

Fig. 5. Matching rate (top 3)

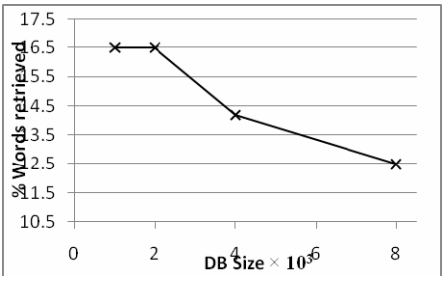

Fig. 6. Percentage of pictograms retrieved

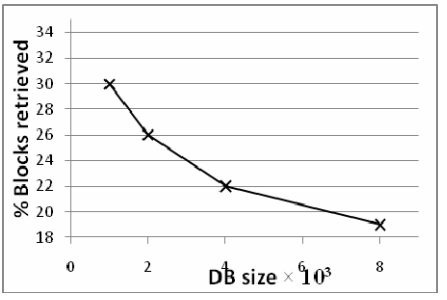

Fig. 7. Percentage of blocks retrieved

the size of the database. Figure 7 shows the percentage of blocks retrieved by the tree search as a function of the database size. Again, since the relative size of the retrieved subset decreases, so does the percentage of blocks brought to memory.

5.3 Evaluation of the Stroke-Based Index

We implemented the proposed stroke-based index and the VUE algorithm [14]. We carried several experiments to evaluate the performance of our proposed index and compare it with the VUE algorithm. Due to the space limitation, we do not show all the results. We asked one writer to produce 200 handwritten cursive words. The same writer then recreated 74 words to be used as search strings. In all the experiments, the stroke location was stored as additional feature as explained in Section 3.5. For the experiments shown here, the value *w* was set at 3 and the value of *s* was set at 1 (each stroke was stored as a separate point.) Since the data used in the experiments were static, we used the Hilbert-packed R-trees [9] as an underlying multi-dimensional index. For data that has dynamic nature (where data can be inserted or deleted at any time), the R-tree [6], Hilbert R-tree [10] or the R*-tree [1] can be used. Node size was fixed at one KByte.

The segmentation algorithm cuts the stroke once it encounters a local minimum. During our experiments, we noticed that some of the resulting strokes are tiny and do not contribute to the final image of the string, and thus considered noise. These tiny strokes can be produced simply by pressing or raising the pen. These strokes not only increase the size of the database but might also adversely affect the retrieval performance. We filtered out such strokes from both the database and the query strings. We only included strokes whose MBR diagonal is larger than 15 points (where the point is the unit distance in the tablet device.)

Figure 8 compares the search time of our proposed index with the search time of the VUE algorithm for different database sizes. As expected the VUE algorithm time increases linearly with the database size. Our proposed index achieves substantial saving in response time over the VUE. Note that the VUE algorithm is faster than the index for small database (less than 15 words) because of the constant overhead of the R-tree. The saving in time, when using the index, increases with the database size.

We also compared the matching rate of the proposed index and the VUE algorithm. Figure 9 shows the number of times the correct answer (matching rate) is

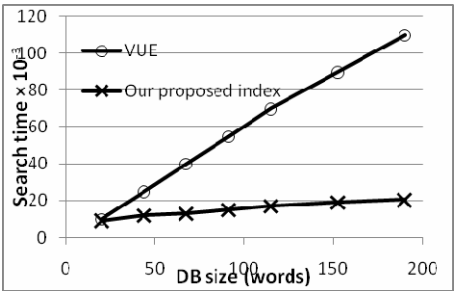

Fig. 8. Response time of our proposed index versus the VUE algorithm

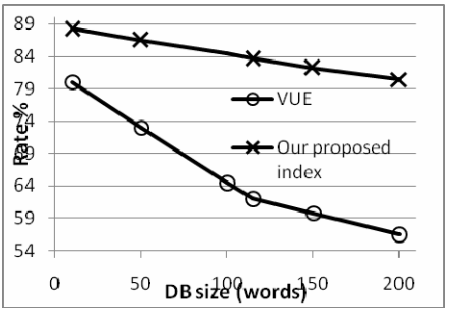

Fig. 9. Matching rate of our proposed index versus VUE algorithm

Table 1. Matching rates for index that uses all 11 features vs, index that uses 6 features only

Voting algo, rank	Matching rate	
	11 features	6 features
first	80	73
top 2	85	82.5
top 3	89	84

ranked among top two for different database sizes. We also carried experiments that show the matching rate when the answer is ranked the first (received highest score) and among the top 5 for different database sizes (not shown for space limitation). The common observation is that the matching rate of our proposed index is consistently higher than that of the VUE algorithm. The improvement in the matching rate is up to 46%.

To evaluate the index when it uses the reduced feature space (as discussed in Section 3.3.), we carried out two sets of experiments, one using the full set of features (= 11). In the second set of experiments we applied the *Karhunen-Loève* transform to a sample of 30 words to calculate the transformation matrix, and then all words in the database were mapped to the new six-dimensional space. The queries were also mapped using the same transformation matrix before searching the tree. Our experiments measured matching rate. We count the number of search words that were ranked first (received the highest score), among top two, and among top three by the voting algorithm. As we see in Table 1, the matching rate is about 84% when reporting strings with the highest three scores. As expected, the matching rate decreased as we used a smaller number of dimensions. The good news is that, although we cut the space required to store a stroke to nearly half, we nevertheless achieved about 93% of the matching power of the index that used all 11 features.

6 Conclusions

We have introduced two new methods for indexing cursive handwriting. The first index works at the word-level and suitable for large database of cursive handwritten

text. While the second index, which works at the stroke level is more accurate but it is also more costly.

The word-level index uses a set of global "word" features that provides an effective way of reducing searching cost. The experimental results showed that the proposed index, which is using R-trees followed by EID clearly outperforms the ED and EID searches. The space overhead incurred by the R-tree is low. The sequential algorithm EID outperforms ED. Another important contribution is the identification of a small set of global features (eight features) that can be used to characterize cursive handwriting.

In the second index, each string is divided into a set of strokes; each stroke is described with a feature vector. Subsequently, the feature vectors can be stored in any multi-dimensional access method, such as the R-tree. A similarity search can be performed by executing a few range queries and by then applying a simple voting algorithm to the output to select the most similar strings. The stroke-level index is resilient to the errors resulting from segmentation errors, such as insertion, stroke deletion, or m-n substitution. Our experiments showed that the extra effort we spent in mapping the data to lower dimensionality space pays off. The stroke-level index achieves substantial saving in search time over the VUE algorithm and improves the matching rate up to 46% over the VUE algorithm. With a sacrifice of less than 10% of the matching accuracy we saved almost half of the space required to represent a stroke.

References

1. Beckmann, N., et al.: The R*-tree: an efficient and robust access method for points and rectangles. In: Proc. of ACM SIGMOD (1990)
2. Jawahar, C.V., Balasubramanian, A., Meshesha, M., Namboodiri, A.: Retrieval of online handwriting by synthesis and matching. Pattern Recognition 42(7) (2009)
3. Roussopoulos, N., Leifker, D.: Direct spatial search on pictorial databases using packed r-trees. In: ACM SIGMOD, pp. 17–31 (1985)
4. Wang, J., Wu, C., Xu, Y., Shum, H.: Combining shape and physical models for on-line cursive handwriting synthesis. International Journal of Document Analysis and Recognition 7(4), 219–227 (2005)
5. Gersha, A., Gray, R.: Vector Quantization and Signal Compression. Kluwer Academic, Dordrecht (1992)
6. Guttman, A.: R-trees: a dynamic index structure for spatial searching. In: Proc. of ACM SIGMOD (1984)
7. Jain, A., Namboodiri, A.: Indexing and Retrieval of On-line Handwritten Documents. In: Proc. of the 7th International Conference on Document Analysis and Recognition, p. 655 (2003)
8. Oda, H., Kitadai, A., Onuma, M., Nakagawa, M.: A Search Method for On-Line Handwritten Text Employing Writing-Box-Free Handwriting Recognition. In: Proc. of the 9th International Workshop on Frontiers in Handwriting Recognition, pp. 545–550 (2004)
9. Kamel, I., Faloutsos, C.: On packing Rtrees. In: Proc. of CIKM (1993)
10. Kamel, I., Faloutsos, C.: Hilbert R-tree: an improved r-tree using fractals. In: VLDB (1994)

11. Zheng, Y., Doermann, D.: Handwriting Matching and Its Application to Handwriting Synthesis. In: Proceedings of the 8th International Conference on Document Analysis and Recognition, pp. 861–865 (2005)
12. Ma, Y., Zhang, C.: Retrieval of cursive Chinese handwritten annotations based on radical model United States Patent 6681044 (2004)
13. Lopresti, D., Tomkins, A.: Pictographic naming. In: Tech. Rep. MITL- TR-21-92, Matsushita Information Technology Lab
14. Lopresti, D., Tomkins, A.: On the searchability of electronic ink. In: Tech. Rep. MITL-TR-114-94, Matsushita Information Technology Lab
15. Rubine, D.: The automatic recognition of gestures. In: PhD thesis, Carnegie Mellon University (1991)
16. Gatos, B., Pratikakis, I., Perantonis, S.J.: Hybrid Off-Line Cursive Handwriting Word Recognition. Pattern Recognition 2, 998–1002 (2006)
17. Al Aghbari, Z., Brook, S.: HAH manuscripts: A holistic paradigm for classifying and retrieving historical Arabic handwritten documents. Journal of Expert Systems with Applications (2009)
18. Wagner, R., Fisher, M.: The string-to-string correction problem. Journal of ACM 21, 168–173 (1974)
19. Varga, T., Bunke, H.: Generation of Synthetic Training Data for an HMM-based Handwriting Recognition System. In: Proc. of the 7th International Conference on Document Analysis and Recognition, p. 618 (2003)