

Rule-based cleanup of on-line English ink notes

Zhouchen Lin^{a,*}, Rongrong Wang^b, Heung-Yeung Shum^a

^aMicrosoft Research, Asia, Zhichun Road #49, Haidian District, Beijing 100080, PR China

^bFudan University, Handan Road #220, Shanghai 200433, PR China

Received 24 November 2004; accepted 9 August 2005

Abstract

Recently, many pen-based devices have enabled people to input digital ink naturally. Often, there is smear and correction when writing. This not only makes the document dirty and look unpleasant, but also affects the handwriting recognition when recognition is called for. As the first paper to address the ink cleanup problem, we present our ink cleanup system that removes the smear and correction, so that the document becomes cleaner and more legible and the handwriting recognition rate could also be improved. The algorithms are rule-based and are capable of dealing with the most common cases that may happen during writing, including self-overtracing of a single stroke, inter-overtracing between strokes, correction, touch-up, insertion and wrong writing order. Experimental results show that our system is effective in cleaning the ink note and is promising in increasing the recognition rate as well.

© 2005 Pattern Recognition Society. Published by Elsevier Ltd. All rights reserved.

Keywords: Document analysis; Document and text editing; Handwriting analysis; Handwriting recognition

1. Introduction

Handwriting is the most important way to expand human memory and facilitate communication. With the increase of the computing power of computers, people are relying more on handwriting recognition technologies [1–6] to convert documents into texts and graphics so that the documents can better be stored, shared, retrieved, and so on. Recently, the flourish of mobile working, particularly the emergence of PDA, Tablet PC, etc., are enabling people to produce more and more handwritten words and even ink documents. When writing on such devices, it is common that people write something erroneous and then fix them. The resultant smear and correction not only make the words or document dirty and look unpleasant, but also decreases the recognition accuracy if such function is called for. Although both off-line and on-line handwriting recognition have been studied by many scholars during the past decades [3–6], to our best

knowledge, we have not seen that the handwriting cleanup problem is addressed in the literature. Usually, the preprocessing of an on-line handwriting recognizer only includes data smoothing, signal filtering, dehooking and break corrections [4], etc. This may be partly because the cleanup problem is not very important in the past as there is still room for improving the recognition accuracy for those “clean” words. Another reason may be that in the past the smeared or corrected words were relatively few so that people did not take them seriously. The third reason may lie in the belief that the training process of the handwriting recognizer can automatically deal with the smearing and correction as long as these cases happen in the training samples. Unfortunately, such a belief is just a misconception because the deteriorated words account for relatively small portion of the training samples and thus will have little effect on the training. The fourth reason may be that cleanup is a much less severe problem in off-line handwriting recognition and people may convert on-line recognition to off-line recognition to bypass this problem. However, the accuracy of off-line recognition is usually lower than that of on-line recognition [3,4]. Nowadays, the increasing demand on higher recognition accuracy disallows such conversion. Therefore, we have

* Corresponding author. Tel.: +86 10 62617711x3143;
fax: +86 10 88097306.

E-mail addresses: zhoulin@microsoft.com (Z. Lin),
rrwang@fudan.edu.cn (R. Wang), hshum@microsoft.com (H.-Y. Shum).

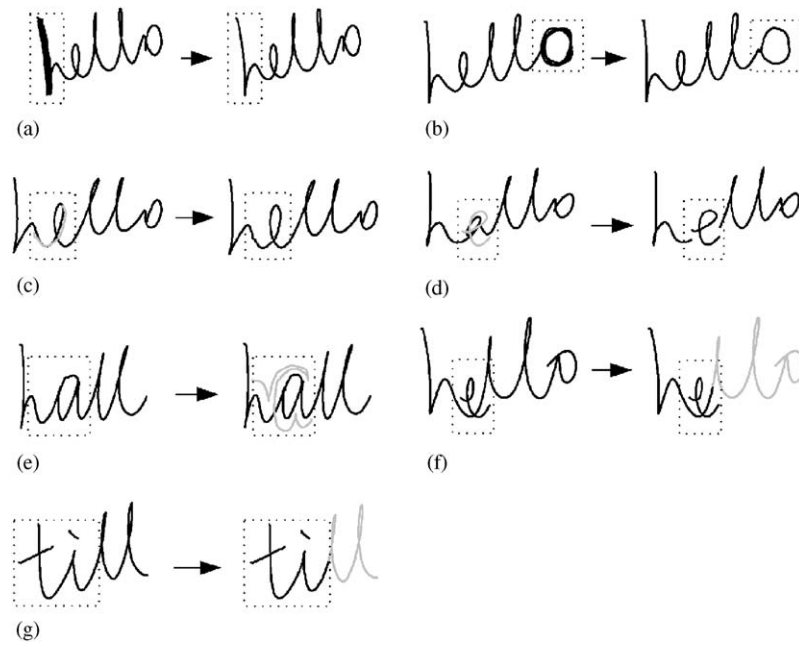


Fig. 1. The cases dealt with by our ink cleanup system and the corresponding desired results. (a) and (b) Self-overtracing by folding and looping, respectively. After cleanup, the overtracing parts are simplified. (c) Inter-overtracing. The overtracing parts are merged. (d) Correction. The background “a” is replaced by the foreground “e”. (e) Touch-up. Two strokes become one stroke, with the writing order shown in grey curve on the right. (f) Insertion. “e” is inserted between “h” and “llo”. The single stroke for “hlllo” is broken into two strokes (black and grey stroke on the right). (g) Late stroke. The t-bar and i-dot are rearranged near to their stems. In the example, the t-bar is arranged before its stem because its stem is the first downward stroke piece of the stroke. And the i-dot is right after its stem. Note that the stroke containing the i-stem is broken into two strokes.

to consider the recognition with degraded words. And our another motivation is to make the on-line ink documents more readable. In this paper, as primary investigation, we seek to handle these two issues in a unified system.

Our system adopts rule-based approaches because the data collection is a hard problem. Although theoretically the deteriorated words can be collected from large amount of data set, detecting the words that we want is not trivial. For example, collecting words with incorrect writing order of strokes may require the visualization of the stroke order and heavy human examination. Moreover, data labeling is also hard, e.g., specifying what the cleaned strokes should be and the correct writing order for the cleaned words is not easy. Due to such difficulties, we have to apply complicated rules that are summarized from our observations to clean the “bad” words. As a result, our rule-based algorithms can only work on English handwriting. In addition, as we conceive our system as the preprocessing of general handwriting recognizers, we do not utilize any recognition results to assist our processing. Therefore, we can only analyze the geometric shapes of strokes and their relationship in detail. Finally, as the number of “bad” words are usually less than “good” words, we have to make the algorithms conservative so that those “good” words will not be processed.

Our system is designed to deal with the following kinds of word quality degradation (Fig. 1) that we believe are the

most common when writing English words or documents:

- Self-overtracing, i.e., the folding and looping of a single stroke. The “duplicated” parts of a stroke will be simplified (Figs. 1(a) and (b)).
- Inter-overtracing, i.e., the overlapping of two strokes. The two strokes will be merged into one (Fig. 1(c)).
- Character correction, i.e., the replacement of characters. Some characters will be replaced by others that are written later at the same position (Fig. 1(d)).
- Touch-up, i.e., changing a character by adding a short stroke. The two strokes will be merged and the writing order within the merged stroke may be rearranged (Fig. 1(e)).
- Insertion, i.e., adding a missing character between two characters that are already written. The strokes will be inserted at their intended position so that the time order corresponds to their horizontal order (Fig. 1(f)). When the two characters around the missing character are written in one stroke, the stroke will be broken at a specific point.
- Late stroke, i.e., the dot and the bar of “i”, “t”, and so on, are not written right before or after their stems (Fig. 1(g)). The late strokes will be rearranged so that the temporal order complies with the spatial order. Their stems may be severed if they are connected to other characters.

The rest of this paper is organized as follows. Section 2 describes the pre-processing of the cleanup algorithms and

introduces several global features used in cleanup. In Section 3 the main steps of the cleanup algorithms are described in detail, including self-overtracing cleanup, inter-overtracing cleanup, correction cleanup, touch-up cleanup, insertion cleanup and late stroke re-ordering. Experimental results are presented in Section 4. Finally we give conclusion and future work in Section 5.

2. Pre-processing of ink cleanup

Pre-processing of ink cleanup involves dot transformation, extraction of some style features, stroke processing and the generation of indexing bitmap. We assume that the word or line grouping task, i.e., determining which strokes are a word or a line, has been done by other independent system, e.g., the advanced parser developed by Microsoft Research, Asia.

2.1. Dot transformation

Dot transformation, i.e., changing some little circles into short strokes, is unique in our system. We have noticed that some people always draw the dots of “i” and “j” as small circles, which may lead to the failure of the handwriting recognizer. For example, in Fig. 2(a), “ink” is mis-recognized as “pond” if there is no special treatment. To overcome such situation, our system detects such dots according to the sizes and the vertical positions of their bounding boxes and the ratio of their lengths to the sizes of their bounding boxes, and changes them into short strokes that are the diagonals of their bounding boxes. Dot transformation cannot be done after polygonal approximation and equi-distance resampling (Section 2.3) because the small circles will become a single dot.

2.2. Writing style feature extraction

There are several user-dependent features that need to be extracted for the later cleanup procedure.

Dynamic information: The required dynamic information includes: 1. The timestamps of the strokes. 2. The writing speed at each point. 3. The local direction at each point. 4. The local curvature at each point.

Slant estimation: The writing slant will be used in cleanup of correction, insertion and late strokes. To estimate the slant, we first find the downward pieces of the strokes, discard those short pieces, and estimate their principal directions by linear regression (Fig. 3 (a)). We then average the directions of the downward pieces. If the angle between the average direction and the vertical direction is within a range, the average direction is accepted as the writing slant. Otherwise, we simply regard the vertical direction as the writing slant.

Average character width estimation: The average character width W_{char} will be used by the dynamic grouping in correction cleanup so that multiple-stroke characters can be

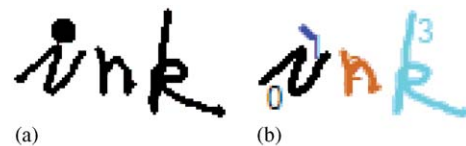


Fig. 2. The necessity of dot transformation. Some people like to write the dots of “i” or “j” in circles, which may cause recognition error. (a) “ink” is recognized as “pond” without dot transformation. (b) After dot transformation, “ink” is correctly recognized as “ink”. Note that the small circle of the i-dot has been changed into a short stroke.

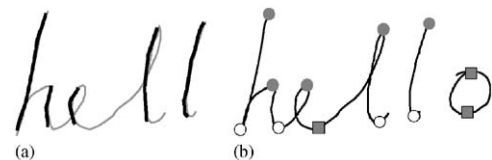


Fig. 3. Downward pieces and the local maxima and minima of the strokes. (a) Black strokes are the downward pieces of the word. (b) Solid dots and hollow dots are the local maxima and minima, respectively. Note that the local extrema indicated by squares are not used because they are of low curvature.

grouped correctly. The estimation procedure is as follows. First, find the local minima that are below the central axis of the writing word. Then sort their x -coordinates and find the x -distance between successive points. Finally, the largest $\frac{1}{4}$ and the smallest $\frac{1}{4}$ distances are discarded¹ and the rest are averaged. The average distance is taken as the average character width.

After all the above features are extracted, a benchmark threshold T_h is taken as the height of the word or the line. In the sequel, all spatial thresholds will be scales of it, and all temporal thresholds are scales of the speed.

2.3. Polygonal approximation and equi-distance resampling

Then, we apply Sklansky’s polygonal approximation [7] to each input stroke and equi-distantly resample them. The following cleanup steps are all processed on the resampled strokes. However, the original strokes are still kept because only those parts that need cleanup will be replaced.

2.4. Indexing bitmap generation

To determine the overlapping parts of the strokes, the system also generate an “indexing bitmap” for each stroke, which records the information of the adjacent stroke points for fast local information collection.

¹As we do not want to distract the reader with too many details, we choose not to present how we deal with the corner cases so that the reader can focus on the general ideas of our algorithms. The rest of this paper also follows this convention.

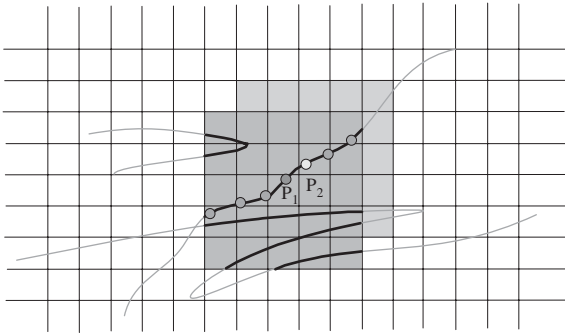


Fig. 4. Indexing bitmap generation. A very important information recorded in each pixel is the number of stroke fragments inside its 3×3 (or 5×5) neighborhood. In the above example, for pixel P_1 , the stroke fragments inside its 5×5 neighborhood are indicated by the heavy curves. Therefore, the number of stroke fragments is 5.

Given the stepsize used in stroke resampling, we first build a bitmap whose pixel size is twice the resampling stepsize of strokes. Then each pixel stores: 1. how many fragments of the strokes inside its 3×3 (or 5×5 , depending on the threshold to define overtracing) neighborhood; 2. the local direction of every point on the stroke fragment; 3. the indices of the strokes; 4. the indices of the points on their corresponding strokes; 5. a flag indicating whether the points are inside the pixel. Note that the number of stroke fragments equals the times that the pen tip passes nearby the pixel, e.g., in Fig. 4 the number of fragments inside the darker 5×5 window is 5.

The information in each pixel is gathered by travelling along the strokes. The required information is filled into all the neighboring pixels that are not the neighbors of the pixel containing the previous point. Take Fig. 4 for example, the information of P_1 contributes to all the darker pixels but that of P_2 only affects the lighter pixels.

3. Ink cleanup algorithms

Our ink cleanup system consists of six engines. Each engine is for one case of cleanup described in Section 1. They are assembled in order. Each engine detects whether the case of interest exists. If not, the engine will change nothing. As we can only analyze the geometric shapes and the relationship between strokes, it is not guaranteed that the cleanup will be correct. Therefore, the cleanup algorithms are made conservative.

3.1. Self-overtracing cleanup

Self-overtracing cleanup is to reduce the parts that a stroke unduly overlaps itself multiple times into a thin stroke or stroke fragment. Note that the backtracing of “l”, “p”, and “m”, etc., should not be processed. Such considerations

poses self-overtracing cleanup a non-trivial problem when there is no recognition.

People may think that changing the stroke into bitmap first and then thinning it by erosion may solve this problem. However, thinning is very sensitive to noise, and recovering the writing order is still an unsolved problem [8–11], which usually involves exponential number of search. Moreover, if there is no special treatment involved, those backtraced “l”, “p”, and “m”, etc., may very likely be mis-processed. Another possibility may be the principal curve [12], which seeks the middle axis of a cluster of point. However, the current available algorithm [12] is too computationally expensive. Therefore, we choose to pose some constraint and process in a different way. We assume that:

- (1) The overtracing part is nearly completely occupied by ink strokes. This avoids some characters like “m” be detected as self-overtracing when they are written compactly. In Fig. 5, cases (a) and (b) will be processed, while cases (d) and (e) will not be processed because there is much space that is not occupied by the strokes.
- (2) The local direction of stroke implies the direction of the cleaned stroke. In Fig. 5, case (c) is valid self-overtracing, while case (d) is not because the local direction does not comply with the global direction.

The flow of self-overtracing cleanup is shown in Fig. 6. We first use the indexing bitmap to detect the self-overtracing parts, i.e., classify the stroke points into overtracing points or non-overtracing points. For each point P of the stroke, suppose the pixel containing P is X and F^- is the set of stroke fragments that are recorded in X but P is not on them. We count the number M of how many fragments in F^- that contain a point whose local direction is nearly the same or inverse to that of P . Mark P as an overtracing point if:

- (1) $M \geq 2$, or
- (2) $M = 1$ and the local direction is nearly the same as that of P , or local direction is nearly inverse to that of P but not close to vertical. The latter condition prevents the backtracing of “l” and “p”, etc., from being processed.

Then we have a sequence of states indicating whether the corresponding point is an overtracing point or not. To be robust, short sub-sequences should be flipped.

Next, we compute the average stroke piece for each sequence of overtracing point. For each non-visited (we will define what is “visited” shortly) overtracing point P , define its searching line L as a line passing P and with direction D that is usually perpendicular to the local direction of P if the stroke does not turn abruptly at P . However, if the stroke turns sharply at P , then D should be modified as the local direction at P . Fig. 7(a) illustrates the dependence of D on the local curvature.

Then starting from P , find, in both directions of the searching line L , the next pixel that the line passes and

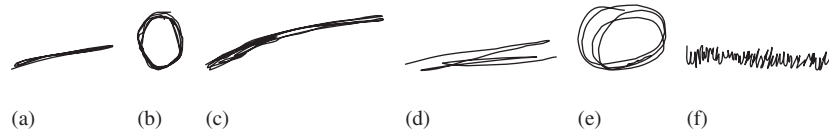


Fig. 5. Supported and non-supported self-overtracing cases. The left three cases are supported, while the right three cases are not.

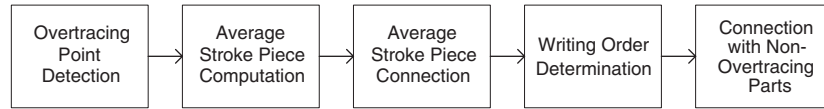


Fig. 6. The flow chart of self-overtracing cleanup.

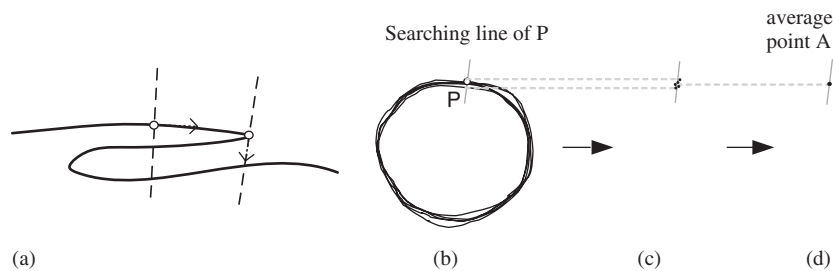


Fig. 7. Compute the average point of overtracing points. (a) The direction of the searching line is dependent on the local curvature. If the curvature is small, the direction of the searching line is perpendicular to the local stroke direction. Otherwise, it is the same as the local stroke direction. (b) For every non-visited point P , collect relevant point along the searching line. (c) The small dots near the searching line are the points to be averaged. (d) The dot is the average point of the dots found in (c) and projected onto the searching line.

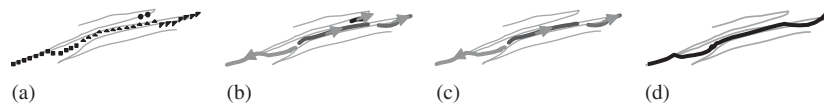


Fig. 8. Generate the average stroke for the overtracing part. (a) Dots of the same shape should be linked to each other successively. (b) Link the average point sequences into average stroke pieces. The arrow direction denotes the direction of the average stroke piece. However, this direction may not be the same as that of the final average stroke. (c) Short average stroke pieces are deleted. (d) Connect the rest average stroke pieces into one stroke so that the total length is minimal.

contains at least one stroke point whose direction of searching line is close to D . The stroke point is not necessary in the current overtracing sequence or non-visited. The search ends when there is no such next pixel. Finally, we average this group of points (the small dots in Fig. 7(c)) and project it onto the searching line L to have an average point A (the dot of Fig. 7(d) and the dots in Fig. 8(a)). Accordingly, the collected stroke points are labelled as “visited”.

The average points are linked to each other successively, in the order they are computed, into an average stroke piece (Fig. 8(b)) until the next stroke point to be processed is visited. Note that with such a linking strategy, a sequence of overtracing points may result in several average stroke pieces (Fig. 8(b)). Then after deleting short average stroke pieces (Fig. 8(c)), the rest pieces are connected into a single stroke by a branch-and-bound searching so that the total length is minimal (Fig. 8(d)).

After the average stroke is generated for each sequence of over-tracing points, it should be linked to the original stroke, before which the writing order within the average stroke must be determined. We have to differentiate two cases of the average stroke: closed and non-closed. If the starting point and the ending point of the average stroke are close to each other, it is considered to be closed. Otherwise, it is not closed. There are several basic criteria to determine the writing order. First of all, the whole average stroke must be traced. Second, the total length should be the shortest. Third, the direction follows the initial direction of the original stroke if the overtracing part is at the beginning or the end of the original stroke.

To determine the writing order, we define P_1 as the ending point of the previous non-overtracing part, P_2 the starting point of the next non-overtracing part, and L_1 and L_2 be on the average stroke which are nearest to P_1 and P_2 ,

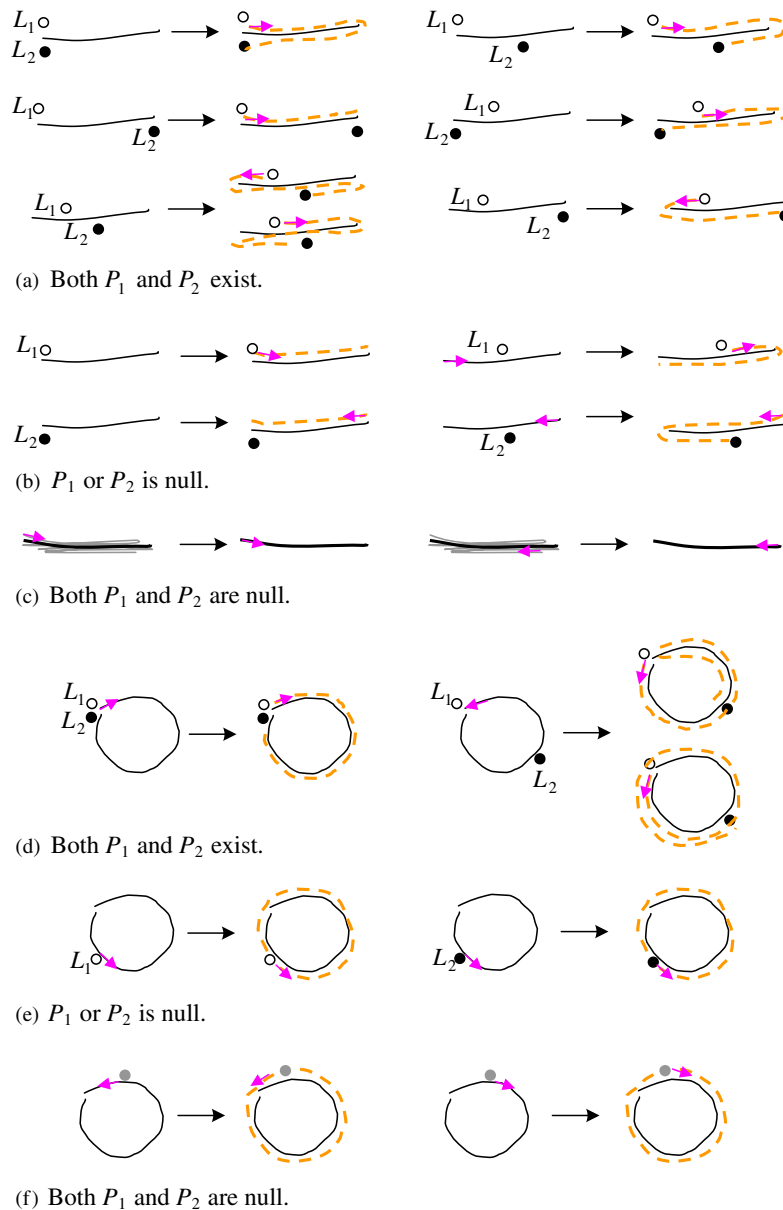


Fig. 9. The rules to recover the writing order, indicated by the dashed lines, in the average stroke. The arrows indicate the direction. (a)–(c) The rules for non-closed average strokes. (d)–(f) The rules for closed average strokes. (c) and (f) When P_1 and P_2 are both null, assign the writing direction as the initial direction of the original stroke and the starting point as the starting point of the original stroke.

respectively. Note that P_1 and P_2 could be null. In this case, L_1 or L_2 is also null. The rules of recovering the writing order within the average stroke depend on whether the stroke is closed and whether P_1 and P_2 are null. The details are illustrated in Fig. 9. We are not to explain them literally due to page limit.

After determining the writing order of the average stroke, we then connect the average stroke back to the non-overtracing parts and smooth the junctions and the average strokes. Thus, we have finished the self-overtracing cleanup of a single stroke.

3.2. Inter-overtracing cleanup

Inter-overtracing cleanup is to merge the overtracing parts between two strokes. Currently our system cannot treat the inter-overtracing among more than two strokes because the data structure and algorithm will be much more complex. Moreover, our system only deals with five cases (Fig. 10), which have characteristics in common that there is only one inter-overtracing part between the strokes and among the four end points of the two strokes there are at most two end points that are not on the inter-overtracing part.

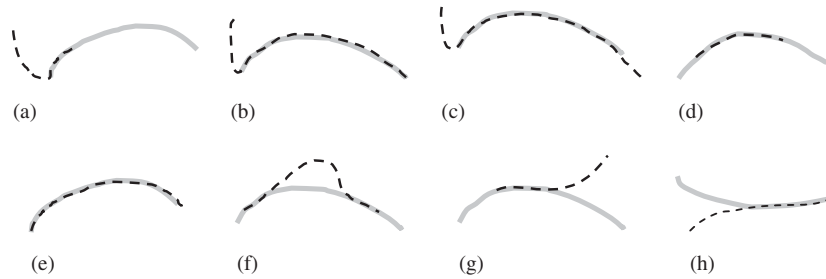


Fig. 10. Supported and non-supported inter-overtracing cases. (a)–(e) are supported while (f)–(h) are not.

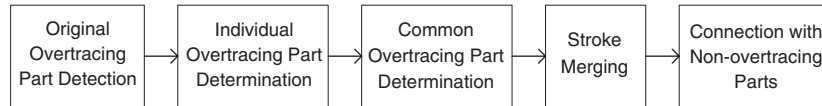


Fig. 11. The flow chart of inter-overtracing cleanup.

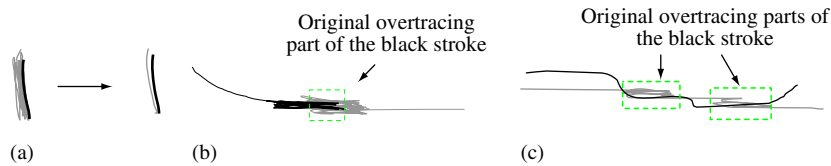


Fig. 12. Inter-overtracing of two original strokes. (a) After the self-overtracing cleanup, the two strokes may not inter-overtracing again. (b) The original overtracing part of the black stroke. In this example, the original overtracing parts are discontinuous in time on the black stroke. (c) The original overtracing parts of the black stroke do not belong to the same sequence of self-overtracing point (see section 3.1), so this inter-overtracing will not be processed.

The stroke pairs in these cases can be merged into a single stroke correctly. For the rest kinds of inter-overtracing, e.g., Figs. 10(f)–(h), we are not very sure about how to determine the writing order after merging the two strokes. Nevertheless, the cases in Figs. 10(f)–(h) may be treated as touch-up if they meet the conditions of touch-up cleanup.

Fig. 11 is the flow of inter-overtracing cleanup. We have to detect the common overtracing part between the two strokes. This is non-trivial because the individual overtracing parts detected on the two strokes individually may not coincide.

The overtracing parts should be detected on original strokes without self-overtracing cleanup. Otherwise, the two strokes may appear non-overtracing after self-overtracing cleanup (Fig. 12(a)).

Given two strokes S_1 and S_2 , where S_2 is later than S_1 . First, we find the original overtracing parts on each stroke. Take S_1 for example, for every point P on S_1 , if the pixel of the indexing bitmap that contains P also contains points on S_2 whose local directions are close or nearly reverse to that of P , then P is marked as an overtracing point on S_1 . Thus, we can obtain a sequence of states indicating whether points of S_1 are overtracing points. Again, to remove possible noise, very short sub-sequences are flipped. Each consecutive sequence of overtracing points form an original overtracing part of S_1 (Fig. 12(b)). If S_1 has several original overtracing parts that do not belong to the same sequence of self-

overtracing points² (Fig. 12(c)), then it is not the case that the system is designed to deal with.

We then find the individual and common overtracing parts of each stroke. Suppose the self-overtracing-cleaned strokes of S_1 and S_2 are S_1^c and S_2^c , respectively. For S_1 , its individual overtracing parts are obtained from projecting, by finding the nearest point, the original overtracing parts onto S_1^c , as shown in Fig. 13(a). For each stroke, its common overtracing part is defined as the union of its individual overtracing parts and the projection of the individual overtracing parts from the other stroke (Fig. 13(b)). If the common overtracing parts on one of the two strokes are not connected to each other (case (f) of Fig. 10), then the inter-overtracing is not supported by our system and we just leave them alone.

Finally, we merge the common overtracing parts of the two strokes. We first judge which case of Fig. 10 happens in the current inter-overtracing, based on the number N of the end points that are not on the common overtracing part.

- For $N > 2$ (case (g) or case (h)), they are unsupported inter-overtracing cases.
- If $N = 0$, this means that S_1^c and S_2^c are completely overlapping (case (e)). Let the curve length parameterization

²Please refer to Section 3.1 for the concept of sequence of self-overtracing points.

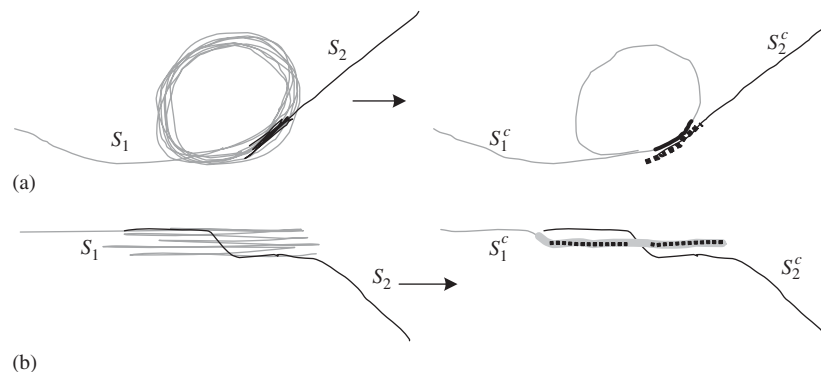


Fig. 13. Get the individual and common overtracing parts of strokes. The strokes on the left of the arrow are the original strokes S_1 and S_2 . Those on the right are the self-overtracing-cleaned strokes S_1^c and S_2^c , and the solid and dashed thick lines are the individual overtracing parts of S_1 and S_2 , respectively. (a) The individual overtracing part is the projection of the original overtracing part onto S_1^c and S_2^c , respectively. (c) The common overtracing part of stroke S_1 is the union of its individual overtracing part (solid thick line) and the projection of the individual overtracing part from the other stroke (dashed thick lines).

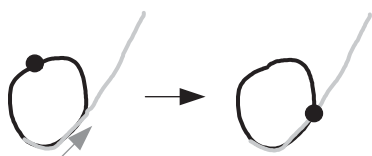


Fig. 14. If one of the inter-overtracing strokes is a loop, its starting point (the heavy dot) should be moved to the ending point of the common overtracing part. Otherwise, the common overtracing part will become self-overtracing part of the merged stroke.

of the two strokes be $S_1^c(t)$ and $S_2^c(t)$ ($0 \leq t \leq 1$), respectively, then the merged stroke is $(S_1^c(t) + S_2^c(t)) / 2$.

- For cases (a)–(d), merge the common overtracing parts using linear combination as case (e) and then concatenate it with the original non-overtracing part.

Note that the direction of S_2 might be reversed so that the direction of its common overtracing part is the same as that of S_1 . Moreover, if one of the strokes is a loop, then its starting point should be moved to the ending point of the common overtracing part so that there will be no self-overtracing in the merged stroke (Fig. 14). After connecting the merged stroke to the non-overtracing part, the junctions and the merged stroke are smoothed, and the timestamp of the new stroke is assigned to be the same as that of S_1 .

3.3. Correction cleanup

Correction cleanup is to use the characters written later (“foreground” character) to replace those written “under” it (“background” character). The main difficulty is to decide, using no recognition information, which strokes form the foreground character and which stroke pieces form the background character. It consists four key steps, as shown in Fig. 15.

3.3.1. Dot and bar binding

This step binds the dots of “i”, “j” and bars of “t” and “f”, etc., with their stems, so that these characters can be made complete. Otherwise, some problems may occur. For example, after stroke replacement the dot of “i” may remain if it is the background character. Or the bar of “t” may be removed as background if people write t-bar first and the t-stem next. Unlike the late stroke re-ordering, the timestamps of these short strokes are unchanged.

To detect the dots and bars, we check the size and shape of the strokes, and test whether the slant projection, i.e., projection along the writing slant of the word, of these short strokes overlap with a downward stroke piece, which may be their stems. Finally, the relative vertical position and the intersection relationship between the short strokes and their stem candidates are tested.

If a short stroke is above its stem candidates and has no intersection, then it should be a dot. If the dot has only one stem candidate (Fig. 16(a)), then the dot is bound with the stem. Otherwise, the dot may have several stem candidates (Fig. 16(b)) and it should be bound with the candidate right on the left of the dot.

If a stroke is close to a horizontal bar and intersects its stem candidates at moderate height and the intersection points are not close to its ends, then it should be a bar, and will be bound with all the unbound stems. Note that it is possible that a t-bar may have several stems (Fig. 16(c)). In this case, the bar should be divided by the number of the stems so that when replacement occur, only part of the bar is removed.

3.3.2. Stroke grouping

This step groups the strokes that are arranged in time order into possible characters by the standard dynamic programming, which is a perfect algorithm to combine local optima to get a global optimum. To apply dynamic programming to stroke grouping, we have to define the probability $p(i, j)$ of

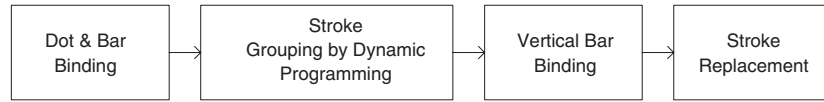


Fig. 15. The flow chart of correction cleanup.



Fig. 16. Examples of dots and bars and their stem candidates. (a) A dot has only one stem candidate. (b) A dot has two stem candidates. (c) A bar has two stem candidates. It should be divided into two sub-bars in order to be bound with the two stems.

the stroke sequence $(i \sim j)$ being a correct grouping. For a stroke sequence $(i \sim j)$, we first compute the following features.

- Width feature $f_w(i, j)$:

$$f_w(i, j) = \begin{cases} 1 & \text{if } n = 1, \\ \exp(-|w/W_{char} - 1|/\sigma_{w,1}) & \text{if } n > 1 \text{ and } w > W_{char}, \\ \sigma_{w,2} + (1 - \sigma_{w,2}) \times w/W_{char} & \text{if } n > 1 \text{ and } w \leq W_{char}, \end{cases}$$

where $n = j - i + 1$ is the number of strokes in the sub-sequence $(i \sim j)$, w is the total width of the stroke group (the width of their slant bounding box, whose vertical edge is in the direction of the writing slant), W_{char} is the average character width (Section 2.2), and the parameters $\sigma_{w,1} > 0$ and $\sigma_{w,2} \geq 1$ control the steepness of the feature function. Obviously, this feature function encourages grouping under W_{char} and the width of the grouped strokes should be as narrow as possible.

- Height feature $f_h(i, j)$:

$$f_h(i, j) = 1 + h/T_h,$$

where h is the height of slant bounding box of the stroke group, and T_h is the height of the handwriting word or line (Section 2.2).

Note that the dots and bars detected in previous step are still taken into consideration when computing the bounding boxes of the sub-groups. However, they are not considered when counting the number of strokes with each sub-group. The final fuzzy function is $p(i, j) = \lambda f_w(i, j) f_h(i, j)$, where $\lambda \in (0, \frac{2}{3})$. The smaller the λ is, the more encouraged the stroke grouping is. With $p(i, j)$, the probability $P(i, j)$ of the optimal segmentation of stroke sequence $(i \sim j)$

can be computed by the following recursion:

$$P(i, i) = p(i, i),$$

$$P(i, j) = \max(\{p(i, k) \times P(k + 1, j) | i \leq k < j\} \cup \{p(i, j)\}), \quad j > i.$$

The optimal segmentation can be found by back-tracing the indices k 's that produces $P(i, j)$ in each recursion.

3.3.3. Vertical bar binding

After stroke grouping, a major problem is that the vertical bars of “B”, “D”, “E”, “F”, “K”, “H”, “M”, “N”, “P”, and “R”, etc., may not be grouped by dynamic grouping because the width of these capital letters are often larger than W_{char} . Vertical bar binding then binds the vertical bars with appropriate capital letters.

If the length of an isolated vertical bar is longer than a threshold, it is bound with its neighbor, as described below:

- If there are two temporally neighboring ungrouped vertical bars, check if the stroke just before or after them is a horizontal bar between them. If so, bind these three strokes together. This is to deal with “H”.
- If the vertical bar intersects its next stroke group or does not enlarge the bounding box of its next stroke group much, then bind them. This is to deal with “K”, “P”, “R”, “B”, “D”, “E”, “F”, etc.
- Otherwise, if the vertical bar intersects its previous stroke group or does not enlarge the bounding box of its previous stroke group much, then bind them. This is to deal with “K”, “B”, “D”, “E”, “F”, etc., that are written in a reversed order.

3.3.4. Stroke replacement

At this stage, we assume that the character grouping has been completed. Stroke replacement then removes the background stroke pieces.

The correction-area is first detected at the level of stroke groups. If the *slant* bounding boxes of two stroke groups overlap, a correction may occur between them. Let S_2 stands for the latter stroke group and S_1 the previous stroke group. Stroke replacement is done as follows:

- If S_2 is a single-stroke group, and its bounding box is flat, and the vertical position of the bounding box is higher than a threshold, then S_2 might be an undetected or unbound bar. Just leave S_2 and S_1 as they are.
- If the width of the *slant* bounding box of S_1 is small enough, replace S_1 with S_2 .

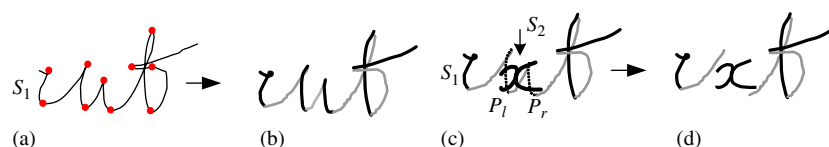


Fig. 17. Procedure of stroke replacement. (a) Find y-min-max points (solid dots). (b) Segment the stroke into pieces. Note that short pieces are merged with longer pieces. (c) Find the overlapped part of S_1 (from P_l to P_r) that should be removed. (d) The overlapped part of S_1 is replaced by the foreground strokes.

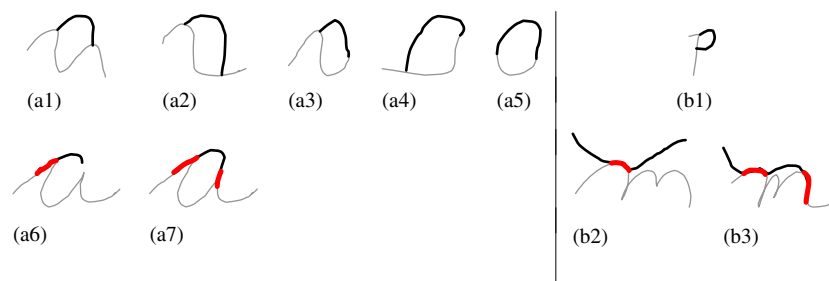


Fig. 18. Supported and non-supported touch-up cases. (a1)–(a7) are supported and (b1)–(b3) are not. (a1)–(a5) touch at least one high-curvature or end point and (a6) and (a7) have only one overlapping part at each end point. (b1) has a vertical stroke, while (b2) and (b3) have overlapping parts that are not at the end points.

- Otherwise find the overlapping part of S_1 and replace it with S_2 .

- (1) First segment S_1 into pieces at y-min-max points or very high curvature points (Fig. 17(a)). Merge the short segments with the pieces right before or after them (Fig. 17(b)).
- (2) Then, starting from the beginning and the end of S_1 , find the first piece P_l and the last piece P_r whose slant bounding boxes overlap that of S_2 , respectively (the dashed lines of Fig. 17(c)). Delete the part of S_1 from P_l to P_r (Fig. 17(d)).
- (3) Finally, delete the remaining dots or bars if their stems are deleted.

- (3) At least one of the touching points on the touched-up stroke is a high-curvature point or an end point (Figs. 18(a1)–(a5)). If only one of the touching point is an end point or a high-curvature point, the stroke piece of the touched-up stroke containing the touching points should not be a near vertical straight line, in order to avoid mistreatment on “p”, “b”, “d”, etc (Fig. 18(b1)).
- (4) The inter-overtracing can only occur near the end points of the touching-up stroke (Figs. 18(a6) and (a7)).

As a result, in Fig. 18, the cases on the left are treated, while those on the right are not supported.

After the touch-up strokes are detected according to the above criteria, where the inter-overtracing parts can be detected using the indexing bitmap, we try to merge them into a single stroke with correct writing order. There are two cases to deal with: 1. there are two touching points so that the touch-up parts form a loop; 2. there is only one touching point.

We first find the touching points. Let S_1 and S_2 be the touched-up and touching-up strokes, respectively. The touching point(s) on S_1 are where the two strokes begin to diverge. Then we find the point(s) C_1 and C_2 which are within a distance from the touching points and are either with the highest curvature or the end points of S_1 .

If there are two touching points, we find the points P_1 and P_2 on S_2 that are closest to C_1 and C_2 , respectively. Then the sub-stroke SS_1 of S_1 between C_1 and C_2 and the sub-stroke SS_2 of S_2 between P_1 and P_2 form a closed loop whose orientation should usually be counter-clockwise *unless* SS_2 is on the right of SS_1 and the stroke piece of S_1 containing SS_1 is a short straight line, because in this case the loop

3.4. Touch-up cleanup

Touch-up cleanup is to merge two strokes that touch each other into a single stroke with correct writing order. For convenience, the stroke written earlier is called “touched-up” stroke and the other stroke is called “touching-up” stroke. Not all cases of two strokes touching each others will be treated because erroneous writing order can occur if features are not salient. Our system only treats touch-up that meets the following constraints:

- (1) The touching-up stroke should be relatively short and slow and does not intersect with the third stroke.
- (2) Both the touching-up stroke and the part on the touched-up stroke between the touching points are relatively simple. At least they should not have self-intersection and fluctuation.

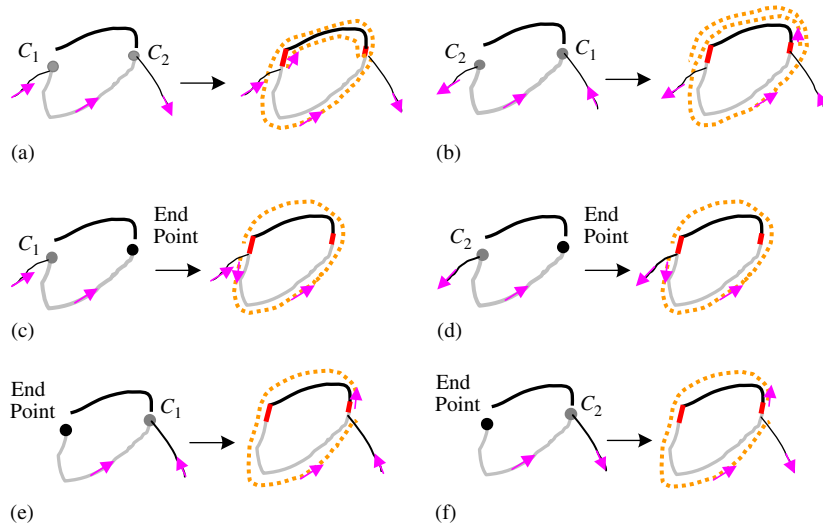


Fig. 19. Six cases of the writing order, indicated by the dashed curves, within the loop when inserting the loop into the broken S_1 , considering appropriate backtracing. C_1 and C_2 are the highest curvature within a distance from the touching points. “End point” is the end point of S_1 . The arrows are the writing directions.

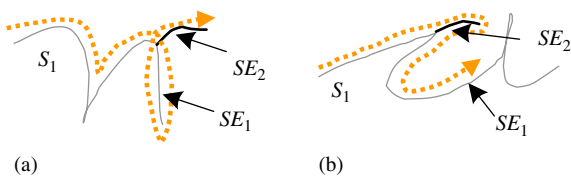


Fig. 20. Two cases of concatenating two touch-up strokes with only one touching point. The writing order is indicated by the dashed curves. (a) SE_1 is short and nearly vertically downward, so backtrace SE_1 and concatenate SE_2 . (b) In the rest cases, SE_2 is backtraced and inserted into S_1 .

may be part of “p” or “b” and hence the loop should be clockwise. Next, we determine the writing order within this loop so that the loop can be connected back with S_1 . There are 6 cases in total (Fig. 19), depending on whether C_1 is before C_2 after re-orientation and whether C_1 and C_2 are the end points. At last, this loop is connected back to the broken S_1 .

If there is only one touching point, we denote by SE_1 the part from the touching point on S_1 to the ending point and by SE_2 the part from the touching point on S_2 to the ending point. If SE_1 is short and nearly vertically downward, then backtrace SE_1 and concatenate SE_2 (Fig. 20(a)). Otherwise, backtrace SE_2 and insert it into S_1 (Fig. 20(b)).

Finally, the junctions are smoothed.

3.5. Insertion cleanup

This step inserts the strokes that are written later at their intended places so that their time order complies with their spatial order. There are two features to decide whether a

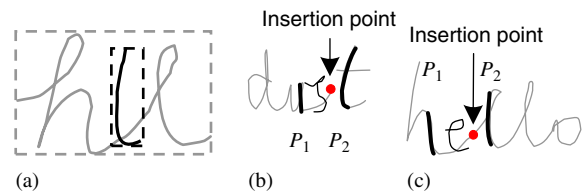


Fig. 21. Insertion cleanup. (a) The black stroke “l” is judged as a to-be-inserted stroke by checking the bounding boxes of it and that of its previous strokes. (b) and (c) Find the pair of downward pieces (the thick lines) containing the to-be-inserted stroke and insert it at the best insertion point.

stroke is a to-be-inserted one.

- (1) It should not be dots and bars for these late strokes will be processed in late stroke re-ordering. These late strokes can be filtered by checking the sizes of their bounding boxes and their vertical positions.
- (2) Its timestamp is either before the stroke *piece* at its left or after the one on its right (Fig. 21(a)).

If a to-be-inserted stroke S is detected, we should find the correct insertion point and then re-order the strokes.

First of all, find the pair of downward stroke pieces P_1 and P_2 so that S is at the right of P_1 and at the left of P_2 , and the distance between them is the shortest in all pairs encompassing S (Figs. 21(b) and (c)). P_1 and P_2 can be null, meaning that there are no appropriate stroke pieces before or after S . If P_2 is null, insertion cleanup is not performed.

Then, collect the sub-sequence of strokes after S so that all of them are between P_1 and P_2 . They together with S form a group of to-be-inserted strokes SS that should all be inserted between P_1 and P_2 . This group is rearranged in the order of left-to-right and bottom-to-top.

Finally, insert SS at the best position among other strokes. If P_1 is null, insert SS before the rest strokes. If P_1 and P_2 are disconnected, then insert SS right after the stroke containing P_1 (Fig. 21(b)). Otherwise, break the stroke containing P_1 and P_2 at their mid-point and insert SS after the mid-point (Fig. 21(c)).

3.6. Late stroke re-ordering

Late stroke re-ordering rearranges the late strokes so that they are right before or after their stems. There are four types of late strokes:

- (1) The dots of “i”, “j”;
- (2) The bars of “t” and “f”;
- (3) The slash of “x”, and the possible dot of handwritten lowercase “z”;
- (4) Single quotation mark “’”;

Since the late strokes of “x” and “z” are actually treated in insertion cleanup, they are not discussed here. The rest kinds of late strokes are detected using the similar method introduced in Section 3.3.1, but more criteria are used. First, a candidate of late stroke should be written later than its stem candidate. Second, the time interval between a late stroke candidate and its previous stroke should not be too large. Third, the writing speed of the late stroke candidate should not be too slow.

All detected late strokes will be re-ordered, i.e., their timestamp will be rearranged. Generally the insertion position of a late stroke is right after its stem. But if its stem is the first downward piece of the stroke containing the stem, this late stroke will be put before the stroke, not after its stem (Fig. 22).

Re-order the dots of “i”, “j” and “’”

Fig. 22. Insertion points for late strokes. Usually the late strokes are inserted right after their stems. However, if the stem is the first downward piece of the stroke containing the stem, then the late stroke should be inserted right before the stroke. In the above example, the dot of “i” is inserted after its stem (black line on the right), while the bar of “t” becomes the first stroke among the three strokes for “think”.

If a late stroke is detected as the dots of “i”, “j” or “’”, it will be re-ordered as below:

- If there is only one stem candidate, then insert the late stroke after the stem. The choice of insertion point is the same as that in insertion cleanup.
- If there are two stem candidates, check whether the first candidate and its next stroke piece form a circle (to detect handwritten “a” and “o”). If so, insert the late stroke after the second stem candidate.
- If the stem candidates belong to the same late stroke, then insert the late stroke after the stem that is closer to the center of the late stroke.
- Otherwise, insert the late stroke after the stem that is just before its center.

Re-order the bars of “f” and “t”

If a late stroke is detected as the bars of “f” or “t”, we just check whether it has stem candidates. If so, it is inserted after the last stem. Otherwise, it should have been dealt with in insertion cleanup.

4. Experimental result

To evaluate our ink cleanup system, we collect 24 ink notes written in English, all of which are of multiple pages.

Table 1
The performance of ink cleanup algorithm

Writer ID	#Labelled Words	#Reco Fail w/o Cleanup	#Reco Fail w/ Cleanup	#Successful Cleanup	#Failing Cleanup	#Non-effective Cleanup	%Successful Cleanup	%Failing Cleanup
1	301	45	40	14	9	14	31.1%	3.5%
2	191	56	61	5	10	35	8.9%	7.4%
3	284	52	53	10	11	26	19.2%	4.7%
4	251	57	55	10	8	34	17.5%	4.1%
5	215	30	26	8	4	13	26.7%	2.2%
6	319	107	110	17	20	44	15.9%	9.4%
7	332	53	56	9	12	21	17.0%	4.3%
8	389	67	73	8	14	26	11.9%	4.3%
Total	2282	467	474	81	88	213	17.3%	4.8%

#Reco Fail w/o Cleanup refers to the number of words failed to be recognized without cleanup. #Reco Fail w/ Cleanup refers to the number of words failed to be recognized after cleanup is applied. #Successful Cleanup refers to how many words that fail to be recognized without cleanup but are successfully recognized after cleanup. #Failing Cleanup is the number of words that are recognized without cleanup but fail to be recognized after cleanup. #Non-effective Cleanup is the number of words that fail to be recognized irrespective of whether cleanup is applied. %Successful Cleanup is defined as #Successful Cleanup/#Reco Fail w/o Cleanup, which is the percentage of correcting the wrong recognition result. %Failing Cleanup is defined as #Failing Cleanup/(#Labelled Words – #Reco Fail w/o Cleanup), which is the percentage of making the correct recognition erroneous.



Fig. 23. Some examples of the effect of ink cleanup. The first column are input strokes. The second column indicates the temporal order of strokes before cleanup is applied. The third column are the cleanup results. Note that the temporal order is labelled at the beginning of each stroke and the strokes are drawn in different colors. Below each figure is the labelled result (first column) or recognition result (second and third columns).

They are taken from eight different writers, who are told to write in their most natural ways without any explicit constraint. Almost all the notes are written in cursive style. The words are then manually labelled. The testing data set contains 2282 English words in total.

The effect of ink cleanup on recognition is shown in Table 1, where the recognition engine is mshwusa.dll that is released internally on April 29, 2003 and is usually in C:\Program Files\Common Files\Microsoft Shared\Ink if a user is using Windows XP® SP1. From the column of #Successful Cleanup, where they are the numbers of words that fail to be recognized without cleanup but are successfully recognized with cleanup, one can see that the cleanup

algorithm can indeed help recognition. Unfortunately, although we have made our ink cleanup algorithms conservative, it still incur undesired results, i.e., some correct recognition results are made incorrect. And this negative effect is comparable with the positive effect in word counts (see the column of #Failing Cleanup). As a result, the overall recognition rate decreases slightly. The main reason is that the correctly recognized words are much more than those incorrectly recognized. Therefore, even a very small percentage of failing cleanup can bring about considerable number of words. If we look at the percentages listed in the last two columns of Table 1, one immediately sees that the percentage of successful cleanup is much higher than that of failing

cleanup. Nevertheless, more effort should be made to further reduce the rate of failing cleanup.

Fig. 23, shows some examples of ink cleanup, on both recognition and visual appearance. We see that our algorithms do have positive effects on both aspects.

5. Conclusions and future work

In this paper, we present our rule-based ink cleanup algorithms for on-line ink notes. They aim at fixing the smear and correction that often happen during the process of writing so that the ink notes can appear cleaner and more legible. Our system is capable of dealing with the most common handwriting problems that exist in on-line ink notes, including self-overtracing, inter-overtracing, correction, touch-up, insertion and wrong writing order of strokes. If one is interested in improving the document legibility only, s/he may remove the dot-transform, touch-up cleanup, insertion cleanup and late stroke re-ordering as the latter three kinds of cleanup virtually do not change the overall appearance of strokes.

Moreover, our system can also act as the pre-preprocessing of handwriting recognition. As a primary investigation, our current system already has a fairly low rate of failing cleanup and a high rate of successful cleanup. Like other rule-based systems, the performance of our ink cleanup system depends heavily on the exactness and completeness of rules that are found. Statistical methods may also be conceived, but only when the training data, whose collection and labeling is not a trivial task, are available can they be tried.

While enriching and improving the existing rules, we are also considering to remove some conservative assumptions in self-overtracing cleanup, inter-overtracing cleanup, and touch-up cleanup. We are fully optimistic that adding more sophisticated and complete rules to our system will eventually improve the recognition accuracy. This is, to the best

of our knowledge, the first time that ink cleanup problem is seriously studied.

References

- [1] S. Impedovo, L. Ottaviano, S. Occhinegro, Optical character recognition: a survey, *Int. J. Pattern Recognition Artif. Intell.* 13 (5) (1991) 1–24.
- [2] V.K. Govindan, A.P. Shivaprasad, Character recognition: a review, *Pattern Recognition* 7 (1990) 671–683.
- [3] R. Plamondon, S.N. Srihari, On-line and off-line handwriting recognition: a comprehensive survey, *IEEE Trans. Pattern Anal. Mach. Intell.* 22 (1) (2000) 63–82.
- [4] C.C. Tappert, C.Y. Suen, T. Wakahara, The state of the art in on-line handwriting recognition, *IEEE Trans. Pattern Anal. Mach. Intell.* 12 (8) (1990) 787–808.
- [5] T. Wakahara, H. Murase, K. Odaka, On-line handwriting recognition, *Proc. IEEE* 80 (7) (1992) 1181–1194.
- [6] F. Nouboud, R. Plamondon, On-line recognition of handprinted characters: survey and beta tests, *Pattern Recognition* 25 (9) (1990) 1031–1044.
- [7] J. Sklansky, V. Gonzalez, Fast polygonal approximation of digitized curves, *Pattern Recognition* 12 (1980) 327–331.
- [8] W. Huang, G. Rong, Z. Bian, Strokes recovering from static handwriting, *Proceedings of the Third International Conference on Document Analysis and Recognition*, vol. 2, Montreal, Canada, August 1995, pp. 861–864.
- [9] Y. Kato, M. Yasuhara, Recovery of drawing order from scanned images of multi-stroke handwriting, *Proceedings of the Fifth International Conference on Document Analysis and Recognition*, Bangalore, India, September 1999, pp. 261–264.
- [10] P.M. Lallican, C. Viard-Gaudin, A Kalman approach for stroke order recovering from off-line handwriting, *Proceedings of the Fourth International Conference on Document Analysis and Recognition*, Ulm, Germany, August 1997, pp. 519–562.
- [11] H. Bunke, et al., Recovery of temporal information of cursively handwritten words for on-line recognition, *Proceedings of the Fourth International Conference on Document Analysis and Recognition*, Ulm, Germany, August 1997, pp. 931–935.
- [12] B. Kégl, A. Krzyżak, Piecewise linear skeletonization using principal curves, *IEEE Trans. Pattern Anal. Mach. Intell.* 24 (1) (2002) 59–74.

About the Author—Zhouchen Lin received the Ph.D. degree in applied mathematics from Peking University in 2000. He is currently a researcher in Visual Computing Group, Microsoft Research, Asia. His research interests include: computer vision, computer graphics, pattern recognition, statistical learning, document processing, and human computer interaction. He is a member of the IEEE.

About the Author—Rongrong Wang is a postgraduate student of Department of Computer Science and Engineering, Fudan University, China. She is currently a visiting student to Microsoft Research, Asia.

About the Author—Heung-Yeung Shum received his Ph.D. in robotics from the School of Computer Science, Carnegie Mellon University in 1996. He is currently the managing director of Microsoft Research, Asia. His research interests include computer vision, computer graphics, human computer interaction, pattern recognition, statistical learning and robotics. He is a senior member of the IEEE.