Pergamon

Pattern Recognition, Vol. 28, No. 4, pp. 493–512, 1995
Elsevier Science Ltd

0031-3203/95 $9.50 + .00

0031-3203(94)00122-7

A CHINESE-CHARACTER THINNING ALGORITHM BASED ON GLOBAL FEATURES AND CONTOUR INFORMATION*

JENN-YIH LIN and ZEN CHEN†
Institute of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, 300, Taiwan, R.O.C.

(*Received* 23 *November* 1993; *in revised form* 30 *August* 1994; *received for publication* 14 *September* 1994)

Abstract—This paper proposes a method of using run-length coding to perform thinning. First, we construct graphs from characters. The attributes (vertical lines, horizontal lines or points) of each node in the graph are determined according to the node's relationship to the nodes above and below it (we will refer to these relationships as global features) and the black runs within the node. Intersections between two adjacent segments are determined on the basis of the graph constructed and contour information. The thinning algorithm thus employs global features and contour information to produce a more accurate skeleton.

Thinning Run Run-length coding Stroke extraction Skeleton

1. INTRODUCTION

Many character thinning algorithms have been proposed in the past two decades. Suen *et al.*[1] surveyed more than 100 thinning methods and classified them into two groups: methods based on iterative deletion of pixels and nonpixel-based methods. Many methods based on iterative deletion of pixels employ a window operator for thinning. These methods place a 3×3, 5×5, or larger $n \times n$ window onto the image and then use a look-up table to determine whether to retain or delete the center black pixel. Some methods even use a window larger than 5×5 (such as the method which uses a 9×9 window),[2] but they make rough decisions in certain cases to reduce the amount of memory needed and the search time.

In general, the advantages of using a window operator to perform thinning are that this approach is simple and can easily be implemented as a parallel algorithm. The disadvantages are that this method is relatively sensitive to noise and that it is less effective than other methods in processing the cross sections of a character. To overcome this second disadvantage, Suen *et al.*[3] extracted ten different cross sections for further thinning and then substituted the thinning result obtained for the thinning result of the method in reference (4).

One type of nonpixel-based method is to employ run-length coding for thinning. This approach is almost used for character thinning.[5–7] This method divides a character into a number of segments (a segment is composed of several connected black runs). The segments and their connections are then used to determined how the segments will be thinned. In reference (5) the merge and fork relationships between runs were used to convert characters into compressed line adjacency graphs (henceforth, "c-LAG"). If there was a significant change in the width of the runs of a node in the graph, the node was divided into a horizontal stroke and a vertical stroke. Vertical strokes with approximately constant width and nearly collinear centerpoints were denoted "candidates for vectorization". Finally, compound vectorization was used to generate the final result. In reference (6), it was assumed that the width of the lines in a character was approximately constant. Merges, forks and significant changes in width in runs were used to construct graphs from characters. Relatively short nodes were regarded as noise and deleted. The line segments in each node were divided into horizontal strokes and vertical strokes; stroke extraction was then performed to obtain the final result. In reference (7), the line segments in Chinese characters were classified into four primitive types of strokes: horizontal strokes, vertical strokes, up-right-slanting strokes, and up-left-slanting strokes. On the basis of knowledge of the structure of Chinese characters, twenty parameters were derived to distinguish between these four primitive strokes. These parameters were then used in the stroke extraction procedure to determine to what type of stroke each line segment belonged.

One recent paper[8] used run-length coding to perform thinning of objects with similar widths. The main aim of this method is to preserve an x-crossing skeleton. It is assumed that the objects in the image are lines or curves of similar width (say, h). If the length of a

* This study was supported by the National Science Council, Republic of China, under contract number NSC83-0408-E-009-009.

† To whom all correspondence should be addressed.

column (row) run is less than 1.2 h, then the midpoint of that run is used to form the skeleton. If the length of a column (row) run is greater than 1.2 h, the clusters of the "long" runs are located, which are found at line intersections or vertical lines. If they are line intersections, the skeletons of the intersections are found using heuristic rules. The column-wise result and the row-wise result are then combined to obtain the final result.

One disadvantage of using a window operator in preprocessing for Chinese character recognition is that much information can be evaluated more easily before thinning takes place. Important information about connecting strokes and touching strokes, for example, may be completely lost when a window operator is used to perform thinning (see Fig. 1). When run-length coding is used to perform character thinning (characters are decomposed into strokes only), on the other hand, the structure of the original character is preserved. Most Chinese characters are composed of vertical lines, horizontal lines and slanted lines. In theory, using run-length coding to perform thinning should produce acceptable results. The presence of noise, connecting strokes, touching strokes and variations in width along the length of a stroke (in printed characters), however, creates segment combinations of many different shapes and hence greatly increases the difficulty of using run-length coding to perform thinning.

At present, the most common drawbacks associated with using run-length coding to perform thinning of Chinese characters are as follows:

(1) the shape of a segment is evaluated on the basis of the runs containing that segment only (or some cases the nearest run in an adjacent segment). Yet using only local information such as this can easily lead to erroneous results, because completely identical segments appearing in different positions may in fact represent different shapes;

(2) the graph constructed and the boundaries of the character shape are not properly utilized to evaluate the relationship between segments, such as whether two adjacent segments intersect or whether a segment lying on a horizontal line is connected with the right or left end of the line;

(3) the treatment given to regions of intersection is too rough. In Chinese characters there are many different types of intersecting shapes and touching strokes, which may intersect at a single point, along a vertical line, or along a horizontal line. In some cases, even two horizontal lines may touch;

(4) using each node in the constructed graph to represent one stroke in a Chinese character is not a very sound approach. For example, in some cases, a short segment lying above (or below) a horizontal line should be joined to the horizontal line. If this short segment is regarded as an independent line, an erroneous skeleton will be produced (see the discussion of graph modification in Section 4).

This paper will propose a method of using run-length coding to perform thinning. The attributes (vertical lines, horizontal lines or points) of each node in the graph are determined according to the node's relationship to the nodes above and below it (we will refer to these relationships as global features) and the runs within the node. In this way, the correct attributes of identical segments in different positions can be obtained by examining the relationships between the nodes and the nodes above and below them. Intersections between two adjacent segments are determined

Fig. 1. An example of stroke touching.

on the basis of the graph constructed and contour information. This thinning algorithm thus employs global features and contour information to produce a more accurate skeleton.

The remainder of this paper is organized as follows. In Section 2, we describe the process of graph construction. Node attributes and their relationships are described in Section 3; the graph modification process is described in Section 4. In Section 5, the thinning algorithm is presented. In Section 6, experimental results are presented that confirm the effectiveness of the proposed thinning process. Section 7 concludes the paper.

2. GRAPH CONSTRUCTION

A Chinese character comprises one or more disconnected components, each of which is composed of one or more strokes. In practice, the connections between these strokes can be classified into the following types, which we have identified from experiments:

(1) merge: two or more strokes merge into a single stroke [see Fig. 2(a)];

(2) fork: a single stroke forks into two or more strokes [see Fig. 2(b)];

(3) A combination of merges and forks [see Fig. 2(c)];

(4) there is a significant change in width between adjacent strokes [see Fig. 2(d)];

(5) the centerpoints of adjacent strokes are not collinear [see Fig. 2(e)].

Because of the effects of noise, the width of the runs in the stroke types described above may not be constant. Except for the first and the last runs, we require that the changes in the width of the runs remain within a certain threshold.

In the following we will employ the stroke connections set forth above to construct the graphs of a character. In the following, if not specified ohterwise, the term "run" denotes a black run.

2.1. *Features and nodes*

In this paper, every run will be classified as representing one of the following features: starting run, fork, merge, end run, or follower. These features are described in detail below (see Fig. 3):

(1) starting run: a run that is not connected to any other run above it is considered a starting run;

(2) fork: a run that connects to two or more runs below it is classified as a fork;

(3) merge: a run connected to two or more runs above it is considered a merge;

(4) end run: a run that is not connected to another run below it is classified as an end run;

(5) follower: a run that is connected to a single run above it and another below it is classified as a follower.

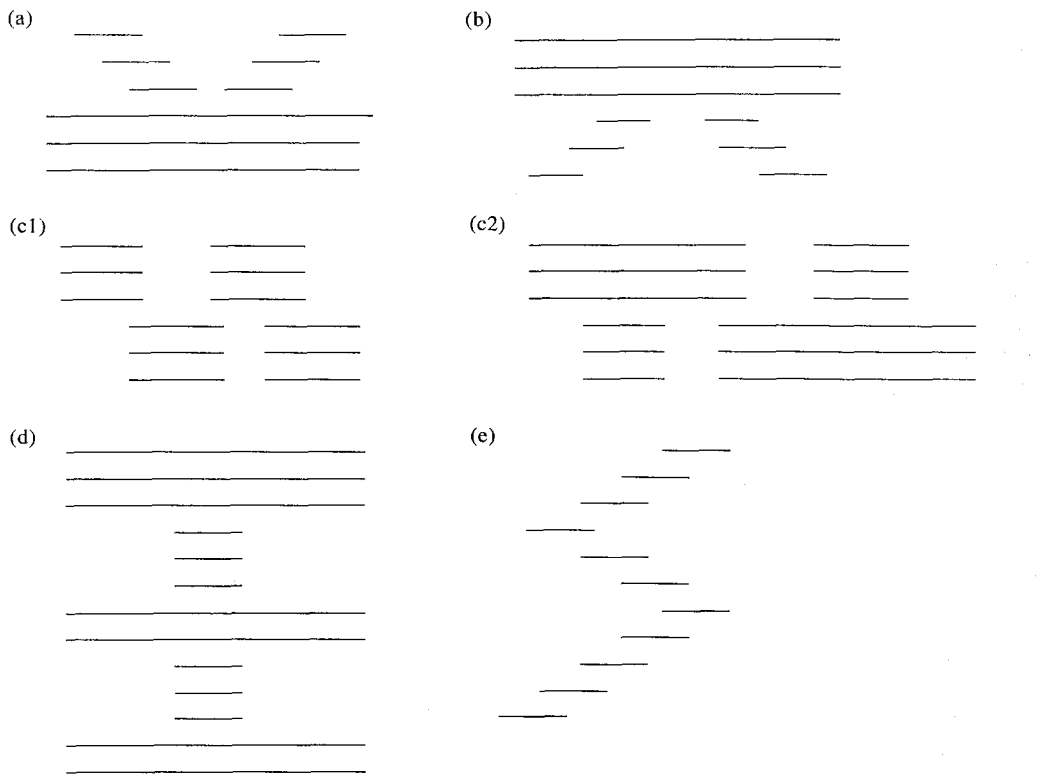


Fig. 2. Examples of connections between two or more strokes. (a) Merge; (b) Fork; (c) A combination of merges and forks; (d) A significant change in width between adjacent strokes; (e) The centerpoints of adjacent strokes are not collinear.

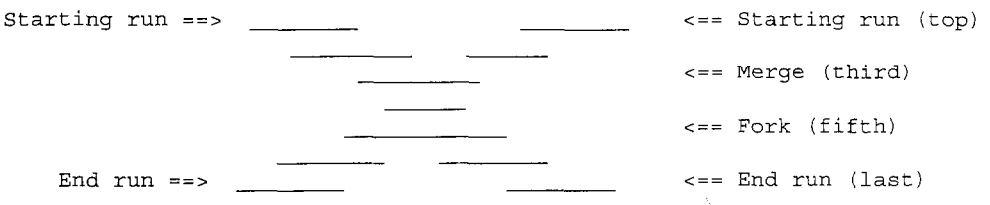


Fig. 3. Types of features (runs for which no feature is specified arc followers).

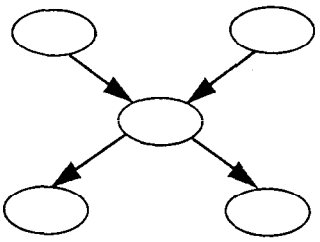


Fig. 4. Graph for Fig. 3.

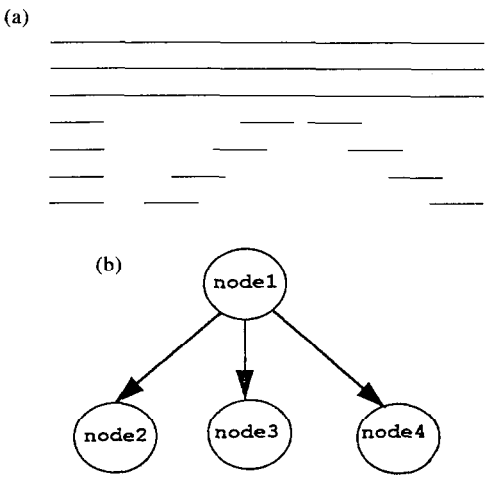


Fig. 5. (a) Run figure; (b) Graph. The node relationships are as follows: node 1 has three nodes below it; node 2 is connected to the left end of node 1, nodes 3 and 4 intersect; node 4 is not connected to the right end of node 1; nodes 2, 3 and 4 each have one node above them.

The runs between any two features (excluding followers) are regarded as nodes. Nodes may have one of three attributes: vertical line, horizontal line or point. (note: below we use line-fitting to perform thinning. For vertical lines, the centerpoints of the horizontal black runs are used to obtain a line equation. For horizontal lines, the centerpoints of the vertical black runs are used to obtain a line equation. For points, fitting is not necessary.) Figure 4 is the graph of Fig. 3.

The information contained in the nodes of a graph includes the following:

(1) the number of runs;
(2) the x-axis and y-axis coordinates of the leftmost point of each run and the width of the run;
(3) the attribute of the node (vertical line, horizontal line, or point);
(4) how many nodes there are above and below the node under consideration and the relationships between these nodes (see Fig. 5);
(5) the value of the label (the runs in different nodes will be labeled with different values).

For example, in Fig. 5, the node relationships are as follows: Node 1 has no nodes above it, but three nodes below it (nodes 2, 3 and 4). Node 2 is connected to the left end of node 1, but not to the right end. Nodes 2 and 3 do not intersect; nodes 3 and 4 intersect. Node 4 is not connected to the right end of node 2. Nodes 2, 3 and 4 each have one node above them (i.e. node 1) and no nodes below them.

2.2. *Processing of features*

During the scanning process, each time a feature is encountered in the scanned image, information about the feature is recorded in the corresponding node. The action of every feature and the information recorded are described below.

2.2.1. *Starting runs.* When a particular run is identified as a starting run, the following procedures are performed:

(1) a new node is established;
(2) the x-axis and y-axis coordinates of the leftmost point of the run and the width of the run are recorded in the node;
(3) the run is assigned a new label;
(4) the value of the label is saved in the node;
(5) the run count for the node is set to 1.

2.2.2. *Followers.* When a run is identified as a follower, the following procedures are performed:

(1) the x-axis and y-axis coordinates of the leftmost point of the run and the width of the run are recorded in the same node as the run above it;
(2) the run is labeled with the same label as the run above it;
(3) the run count for the node in which the run is saved is incremented by one.

2.2.3. *End runs.* When an end run is identified, we need to determine whether the node it is in, is formed of several strokes [see Fig. 2(d) and (e)] and to eliminate noise around the end run.

In order to divide a node into several strokes, we assume every stroke is composed of at least two runs. If a node contains only one run, we merge it into a node above or below it (see Fig. 6), or else we regard it as noise and delete it.

To divide the runs in Fig. 2(d) into five strokes, we take four runs at a time and determine whether the four can be divided into two strokes. Let the widths of runs i, $i+1$, $i+2$, and $i+3$, be W_i, W_{i+1}, W_{i+2} and W_{i+3}, respectively. If equation (2.1) holds, then the runs up to (and including) run $i+1$ form one stroke, and those from run $i+2$ on form a different stroke.

$$(2*W_i < W_{i+2} \quad and \quad 2*W_{i+1} < W_{i+3}) \quad and$$
$$(W_i + thresh1 < W_{i+2} \quad and \quad W_{i+1} + thresh1 < W_{i+3}) \quad and$$
$$(|W_{i+1} - W_i| + |W_{i+3} - W_{i+2}| < |W_{i+2} - W_{i+1}|) \tag{2.1}$$

In equation (2.1), the expression in the first set of parentheses indicates that the width of the wider runs is two or more times that of the narrower runs. The expression in the second set of parentheses is included to prevent the width of the narrower runs from being too narrow (such as a width of only 1 or 2) and hence producing a mistaken result. The expression in the third set of parentheses is included to prevent a single stroke from being divided into two strokes in cases where the width of the end of a stroke is larger than the width of the rest of the stroke (see Fig. 7).

Equation (2.1) applies to cases where the change in the width of successive runs is from narrow to wide. In cases where the width changes from wide to narrow, the following equation is used to divide the runs into separate strokes:

$$(2*W_{i+2} < W_i \quad and \quad 2*W_{i+3} < W_{i+1}) \quad and$$
$$(W_{i+2} + thresh1 < W_i \quad and \quad W_{i+3} + thresh1 < W_{i+1}) \quad and$$
$$(|W_{i+1} - W_i| + |W_{i+3} - W_{i+2}| < |W_{i+2} - W_{i+1}|) \tag{2.2}$$

Once we have used the above equations to determine whether the first group of four runs can be divided into two strokes, we continue to work downward until we have covered all of the runs.

In some cases, the runs in a node should be divided into two strokes, but because of the effects of noise, they may in fact fail to be divided. For example, in Fig. 6, if the width of run 3 is less than twice that of group 2 but more than half that of group 4, then the

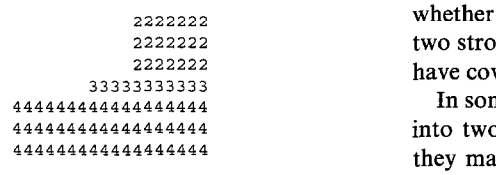


Fig. 6. The run labeled 3 is merged into either group 2 or group 4.

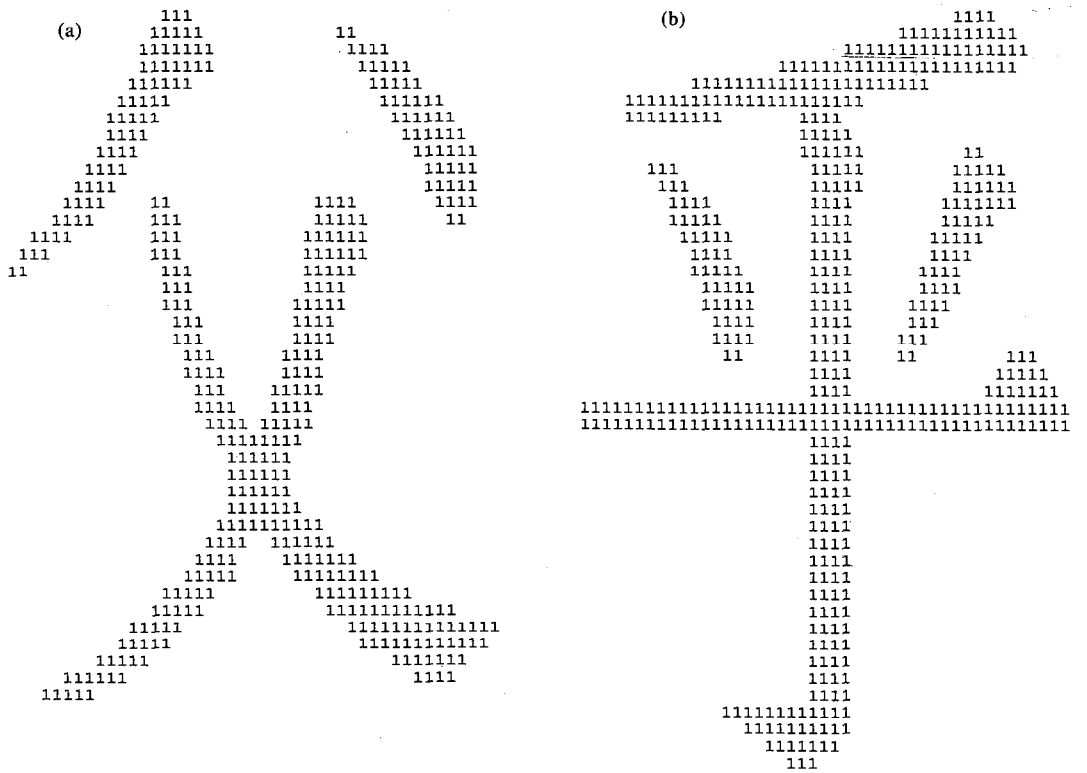


Fig. 7. The width of the end of a stroke is larger than the width of the rest of the stroke.

method described above will fail to divide the runs in Fig. 6 into two strokes. Hence after the above decision method is applied, we check to see whether the number of runs in a stroke is greater than five. If it is, the decision procedure is repeated, this time using groups of five runs instead of groups of four. Suppose that the widths of five runs, from top to bottom, are W_i, W_{i+1}, W_{i+2}, W_{i+3} and W_{i+4}, respectively. Then W_{i+3} and W_{i+4} are substituted for W_{i+2} and W_{i+3}, respectively, in Equations (2.1) and (2.2), and the stroke division procedure is repeated. If one of the equations holds, then the group of runs must be further subdivided, and run W_{i+2} is considered to be a part of the group of longer runs.

To delete noise, at this stage of the procedure we merely delete any nodes consisting of a single run. Any remaining noise is eliminated using the procedure described in Section 4 below.

In summary, when a run is identified as an end run, the following procedures are performed:

(1) if the run can also be identified as a starting run, then the node is regarded as noise and deleted (it is merely a short black run);

(2) if the feature above the run is a fork, then the node in which the current run is located is deleted;

(3) We check whether the node in which the run is located can be divided into two or more different strokes [see Figs 2(d) and (e)]. If it can, then we divide it, save each stroke as an individual node, and reassign the nodes that were above or below the original node so that they connect to the new nodes.

2.2.4. *Forks.* When a fork is identified during the scanning procedure, the following procedures are performed:

(1) The fork node is checked to see whether it can be divided into two or more strokes using a procedure analogous to that described in Section 2.2.3;

(2) in a procedure analogous to that for starting runs, a node is created for each run after the fork and these nodes are inserted below the node that contains the fork run.

2.2.5. *Merges.* When a merge is found, the following procedure is performed:

Step 1. the nodes before the merge are checked and any node with a run count of one is deleted. If after this step is performed no nodes are left above the merge, then the run being scanned is identified as a starting run.

Step 2. if after step 1 there are nodes remaining above the merge, then each node is checked to see if it can be further subdivided into new nodes [see Fig. 2(d) and (e)]. If so, the node is divided and the nodes above and below it are reassigned to connect to the new nodes.

Step 3. (1) if after step 2, the number of nodes preceding the merge is one, then the current run is identified as a follower;

(2) if the number of nodes is more than one, then a new node is created and the node preceding the merge is inserted above this node.

2.2.6. *Graph construction algorithm.* Our graph construction procedure can be summarized by the following algorithm:

Step 1. scan a bilevel image from left to right and top to bottom. When one of the following cases is found to hold, go to step 2.

Case 1. the first pixel scanned is a white pixel; continue scanning rightward until a non-white pixel or the end of the row is reached.

Case 2. the first pixel scanned is a black pixel; continue scanning rightward until a non-black pixel or the end of the row is reached.

Step 2. process the two cases in step 1.

Case 1. in step 1, a white run was scanned. Now check whether a previously labeled run appears above this white run;

(1) if yes, then label each labeled run as an end run;
(2) if no, then go to step 4.

Case 2. In step 1, a black run was scanned. Now check whether a previously labeled run appears above this black run.

(1) if no, then label the currently scanned black run as a starting run;

(2) if two or more labeled runs appear above the current black run, then label the current run as a merge and check whether the rightmost labeled run connected to the run below is a fork run [see Fig. 2(c.1)]. If yes, insert these nodes below the node that the rightmost labeled run is in;

(3) if only one labeled run appears above the current run, then check how many non-white runs in the scanning row are connected to the above labeled run;

(a) if there is only one, then label the currently scanned run as a follower;

(b) if there are two or more, then label the run above as a fork and check whether the rightmost black run connected to the run above is a merge run [see Fig. 2(c.2)]. If yes, insert these nodes above the node that the rightmost black run is in.

Step 3. once a run has been labeled as a starting run, fork, merge, follower, or end run in step 2, execute the processing described in the subsections above for the corresponding type of label.

Step 4. Check whether the scanning process has reached the rightmost pixel in the last line of the image.

(1) if yes, then go to step 5.

(2) if no, then check whether the scanning process has reached the last pixel in the current row;

(a) if no, then go to step 1 and continue scanning rightward;

(b) if yes, then move to the first pixel in the next row, go to step 1 and begin scanning rightward.

Step 5. end.

3. NODE ATTRIBUTES AND NODE RELATIONSHIPS

In this section, we shall discuss two topics:

(1) node attributes: each node is assigned one of three attributes: vertical line, horizontal line, or point;

(2) node relationships: whether the adjacent nodes intersect or whether the nodes above or below the node under consideration are connected on the left end or right end with the node under consideration (see Fig. 5).

3.1. *Node attributes*

Nodes may have one of three attributes: vertical line, horizontal line or point. Unless the ratio of a node's width to its height is very large, in which case the node is definitely a horizontal line, we must examine a particular node's relationship to other nodes above and below it in order to identify the attribute of the node. If we attempt to identify node attributes on the basis of only the run information contained in the nodes, then the risk of error is very high. In the following discussion, we use the parameter "up" to indicate the number of nodes a particular node is connected with above it and the parameter "dn" to indicate the number of nodes a node is connected with below it. Our guidelines for identifying the attributes of nodes are described below.

First we define a series of conditions. (Note: in each of the following, "Cond i" indicates the condition expressed by the words outside the parentheses, while "Cond i(a)" indicates the condition expressed by the words in parentheses.)

Cond 1(a): the location where the node under consideration is connected to the node above (below) is not the first (last) run of the node under consideration (see Fig. 8).

Cond 2(a): the node under consideration is connected to only a single node above (below) it, and the width of the first (last) run of the node under consideration is wider than that of the last (first) run of the node above (below) it plus thresh1.

Cond 3(a): the node under consideration is connected to two or more nodes (assume n nodes) above (below) it, the width of the node under consideration is more than twice its height, and the width of its first (last) run is greater than the sum of the widths of the last (first) runs of all nodes above (below) it plus (n + thresh1).

Cond 4(a): the run count of the node under consideration is less than thresh3 and the widths of the runs are strictly decreasing (increasing) (this criterion can be relaxed to allow several runs to have equal width).

Case 1. up = 0 and dn = 0:

This indicates that the node under consideration is an isolated node. If the ratio of the maximum width of the runs to the height of this node is greater than 1, then the node is a horizontal line; otherwise, it is a vertical line.

Case 2. (up = 1 and dn = 0) or (up = 0 and dn = 1)

this indicates that if the node is not a horizontal line, it is a vertical line. Consider up = 1, dn = 0 as an example:

(a) if the node above the node under consideration is connected to more than one node below it, then the node under consideration is a vertical line;

(b) if Cond 1 or Cond 2 holds, then the node under consideration is a horizontal line;

(c) if neither (a) nor (b) holds, then the node is a vertical line.

An analogous algorithm can be used on up = 0 and dn = 1.

Case 3. up = 1 and dn = 1

This indicates that if the current node is not horizontal line, it is a vertical line.

(a) if the node above the current node is connected to more than one node below it, and the node below the current node is connected to more than one node above it, then the current node is a vertical line.

(b) if Cond 1, Cond 1(a), Cond 2, or Cond 2(a) holds, then the current node is a horizontal line;

(c) if neither (a) nor (b) holds, then the node is a vertical line.

Case 4. (up $\geq$ 2 and dn = 0) or (up = 0 and dn $\geq$ 2)

the current node may be a horizontal line, vertical line, or a point. Consider up $\geq$ 2, dn = 0 as an example:

(a) if Cond 1 or Cond 3 holds, then the current node is a horizontal line;

(b) if Cond 4 holds, then the node is a point;

(c) if neither (a) nor (b) holds, then the node is a vertical line.

Case 5. (up $\geq$ 2 and dn = 1) or (up = 1 and dn $\geq$ 2)

the current node may be a horizontal line, vertical line, or a point. Let us consider up $\geq$ 2, dn = 1 as an example:

(a) if Cond 1, Cond 1(a), Cond 2(a), or Cond 3 holds, then the current node is a horizontal line;

(b) if Cond 4 holds, then the node is a point;

(c) if neither (a) nor (b) holds, then the node is a vertical line.

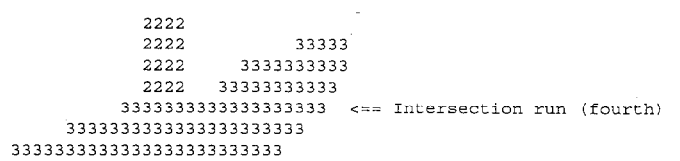


Fig. 8. The node labeled by 2 connects to the node labeled by 3 at its fourth run.

Case 6. up ≥ 2 and dn ≥ 2

the current node may be horizontal line, vertical line, or a point.

(a) if Cond 1, Cond 1(a), Cond 3, or Cond 3(a) holds, then the current node is a horizontal line;

(b) if the widths of the runs in the current node are strictly decreasing followed by strictly increasing and the run count is less than thresh3, then the node is a point (this criterion can be relaxed to allow several runs to have the same width);

(c) if neither (a) nor (b) holds, then the node is a vertical line.

3.2. *Relationships between nodes*

During the thinning process, we need to know whether the left ends (right ends) of strokes formed by two nodes one above the other are connected and whether strokes formed by adjacent nodes intersect. If the strokes do intersect, we need to fit them so that they intersect at one point in order to produce a correct skeleton.

When identifying node relationships, we need only to consider nodes that represent horizontal lines. This is because if nodes that are vertical lines or points are connected to two or more nodes above (below), then the nodes above (below) will definitely intersect.

Before identifying the relationships between nodes, we must first delete any noise above and below vertical lines [see labels 5 and 9 in Fig. 13(c1)].

Suppose node A is a horizontal line, and let node B be the rightmost node above node A to which node A is connected. The following algorithm (algorithm 1) is designed to determine wheter these nodes are connected on the right end (an analogous algorithm can be used to check whether the left end is connected).

Algorithm 1:

Step 1. find the run in node A that has the largest value for its x-axis coordinate and denote it by x_1. Denote its y-axis coordinatee by y_1;
Step 2. find the line equation of the line formed by the rightmost point of each run in node B;
Step 3. obtain the value of x_2 by substituting y_1 into the line equation found in step 2.
Step 4. if $|x_2 - x_1| \leq 2$, then the right end of these nodes is connected; otherwise, they are not connected on the right.

Suppose node A is a horizontal line. If there are n (>1) nodes above node A that are connected with it, the following algorithm (algorithm 2) can be used to determine whether node (i $-$ 1) and node (i) intersect.

Algorithm 2:

Step 1. use line fitting to find the line equation of the line formed by the centerpoints of each run in node (i $-$ 1) and that formed by the centerpoints of each run in node (i);
Step 2. find the intersection of the two lines in step 1 and denote it by (x_f, y_f);
Step 3. find the largest y-axis coordinate among the coordinates of all the runs in node (i $-$ 1) and node (i) and denote it by y_p;
Step 4. if $|y_f - y_p| \leq thresh2$, node (i $-$ 1) and node (i) intersect. Otherwise, node (i $-$ 1) and node (i) do no intersect.

4. GRAPH MODIFICATION

After we have determined the attributes of every node and the relationships between nodes, we further modify and refine the graph in the following ways:

(1) Noise is eliminated as follows: if a vertical line located directly above (below) a horizontal line is found to be unconnected to any node above (below) it and the height of the vertical line is less than half the height of the horizontal line, then the vertical line is deleted.

(2) If a horizontal line is connected to a point above (below) it, the point is deleted [see Fig. 9(c)]. If, after the point is deleted, other points are found, these too are deleted [see Fig. 9(d)].

(3) If two horizontal lines are connected together and the difference in the slope of the lines is negligible, the two lines are merged into a single horizontal line.

(4) To facilitate the processing described in Section 5 below, when a vertical line is connected to two or more vertical lines above (below) it, we add a "point" node between the vertical line and the vertical lines above (below) it (see Fig. 10).

5. THINNING

We have now completed our description of the stroke extraction process and shown how the attributes of each stroke and the relationships between strokes are identified. In this section, we shall describe how the data derived through the processes outlined in Sections 2, 3, and 4 may be used to obtain a character skeleton.

Our approach will be to construct the skeleton of a character by performing line fitting using the attributes of and relationships between nodes in the character graph. For horizontal lines, we shall use the centerpoints of vertical runs for line fitting. For vertical lines, we shall use the centerpoints of horizontal runs for line fitting. The steps in the process are as follows:

Step 1. we begin by fitting all horizontal lines. If the left end or right end of a horizontal line is a connected to a vertical line, then the horizontal line is fitted to half the average line width of the vertical line. The coordinates of the left and right end points of the horizontal line after fitting are then recorded;
Step 2. all points are masked out (fitting is not performed on points), with the centerpoint of each point being used to represent the location of the point. The coordinates of the centerpoints are then recorded;
Step 3. we repeat step 4 through step 7 until all vertical lines have been fitted;
Step 4. we identify a node of a vertical line that has not yet been fitted;

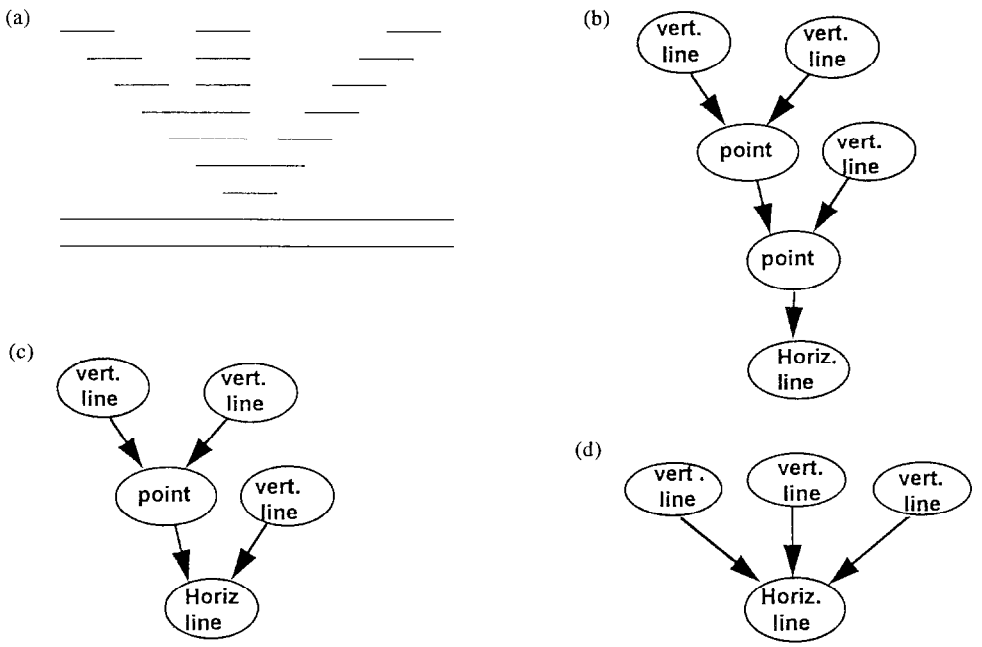


Fig. 9. (a) Run figure; (b) Graph; (c) We delete the point node above a horizontal line node; (d) After (c), if there is still a point node in the horizontal line node, it is also deleted.

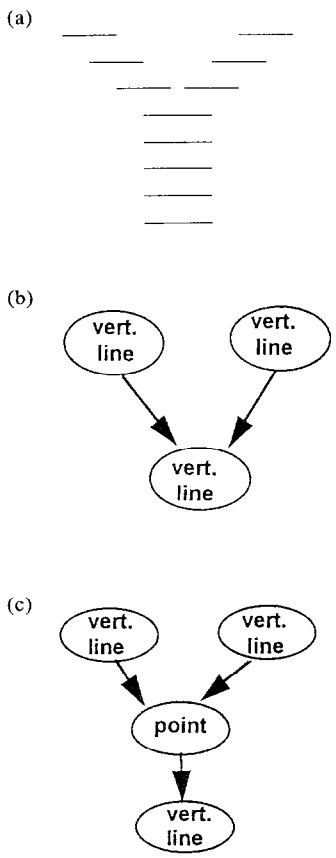


Fig. 10. (a) Run figure; (b) Graph; (c) We add a "point" node between the two existing nodes.

Step 5. for the line identified in step 4, we determine how many collinear vertical lines there are above it and below it. We regard these lines as forming a set, which we will call a line set;

Step 6. we attempt to identify how many points the line set must pass through, in the following way:

if a particular line in the line set is connected to a node whose attribute is "point", then the line must pass through that point;

if a particular line in the line set intersects a neighboring line for which fitting has been completed, then the line must pass through the point of intersection;

if a particular line in the line set is connected to the left end or right end of a horizontal line, then the line must pass through the point where it connects with the horizontal line;

if the very top (bottom) line in the line set intersects another vertical line for which fitting has been completed, then it must pass through the point where it intersects this vertical line;

if the very top (bottom) line in the line set is connected to a horizontal line, and there is another vertical line located above (below) this horizontal line that passes through the horizontal line and intersects with the top (bottom) line of the line set, then if fitting of this other vertical line has been completed, the top (bottom) line in the line set must pass through an end point of this vertical line.

Step 7. after step 6 is completed, we apply the following rules to find the points that form the skeleton of the line set under consideration:

(A) if in step 6 no point is found, then line fitting is used to obtain the line equation of the line set;

(B) if in step 6 only one point is found, then line fitting that must pass through a specific point is used to obtain the line equation for the line set;

(C) if two or more points are found step 6, then working from the top, we search for every two neighboring points and obtain the line equation that passes through these two points.

Step 8. the procedure terminates.

6. EXPERIMENTAL RESULTS

To verify the feasibility of our method, we tested it on printed characters collected and filed by the Telecommunication Laboratories of the Ministry of Transportation and Communications. The highest resolution of the characters is 48*48. We selected a sample of 700 characters by beginning from character 500 in the list and selecting every fifth character until we reached character 4000. The characters in our sample had the following characteristics: there were many different types of touching strokes and intersections; there were cases where a single run formed a complete stroke; and there were cases where the stroke boundaries contained noise. In the test results, correct skeletons were obtained for all but eleven out of 700 characters, so the success rate was 98.43%.

The algorithm is written in Microsoft C language on an IBM PC 486-33. The total execution time for using run-length coding to extract the strokes of 700 Chinese characters (including reading data from disk) is 512.83 s, or on average, 0.73 s for each character. On the other hand, total computer time including line fitting the extracted strokes to produce the skeletons for the 700 Chinese characters is 821.31 s, or 1.17 s for each character.

During the thinning stage, in some cases we had to force lines to pass through certain points, causing some of the thinning results to become slightly curved [see Fig. 11(a)]. This is because the center point of a wider run may shift to one side dramatically, yet after thinning

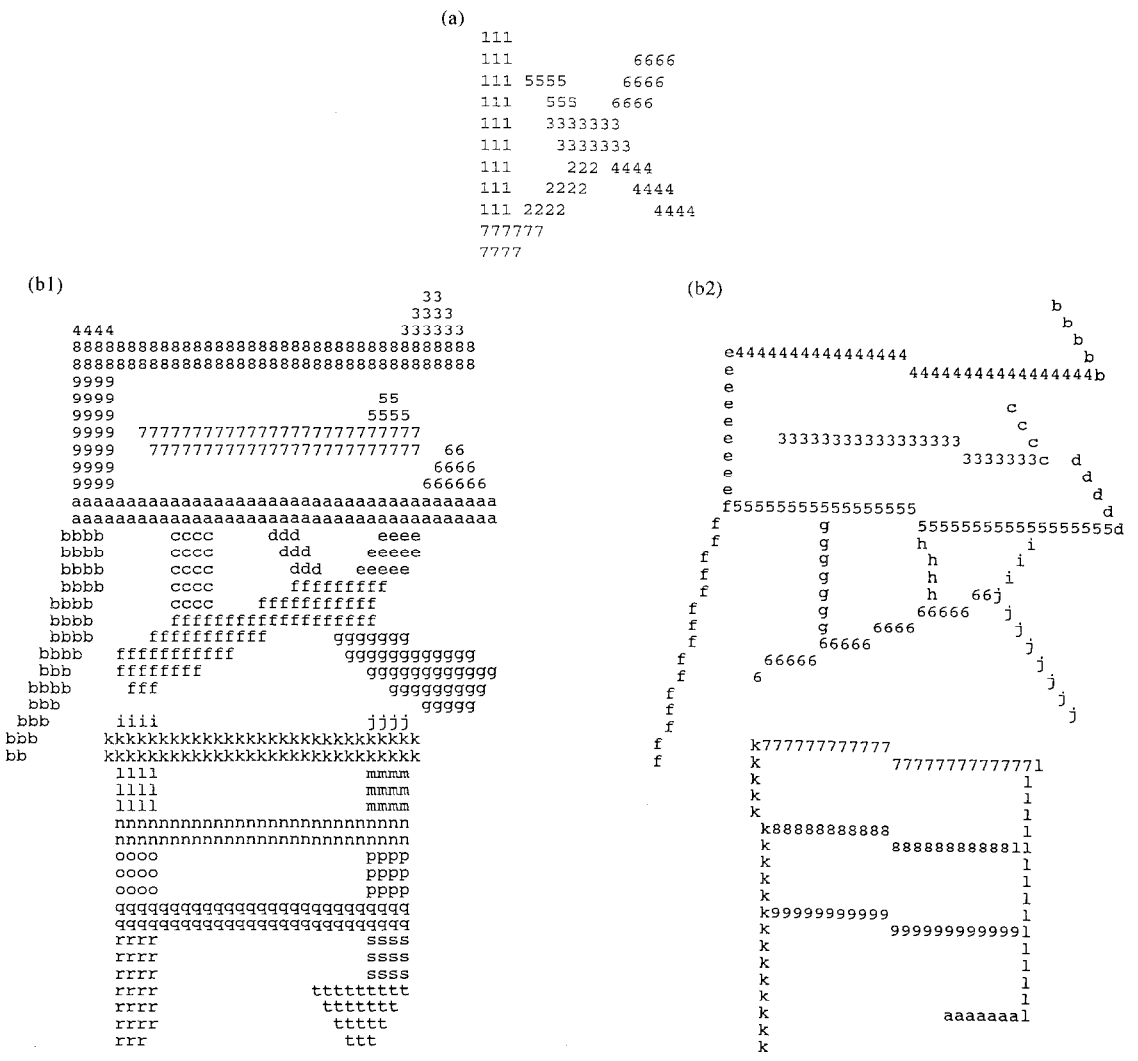
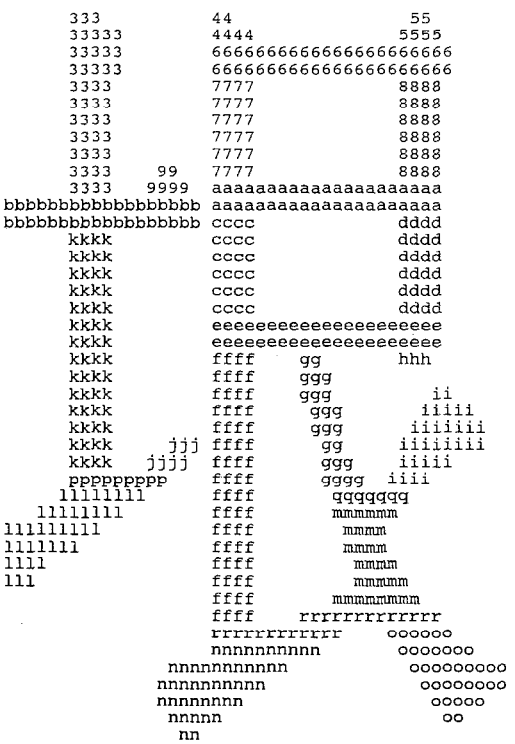
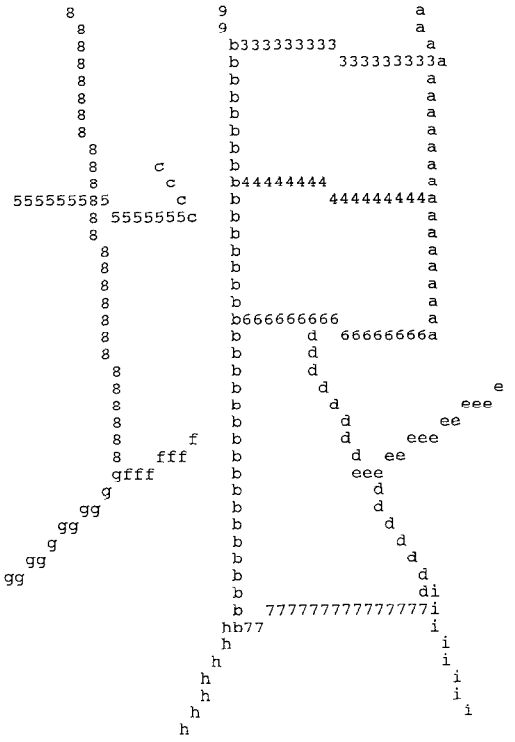


Fig. 11 (*Cont.*)

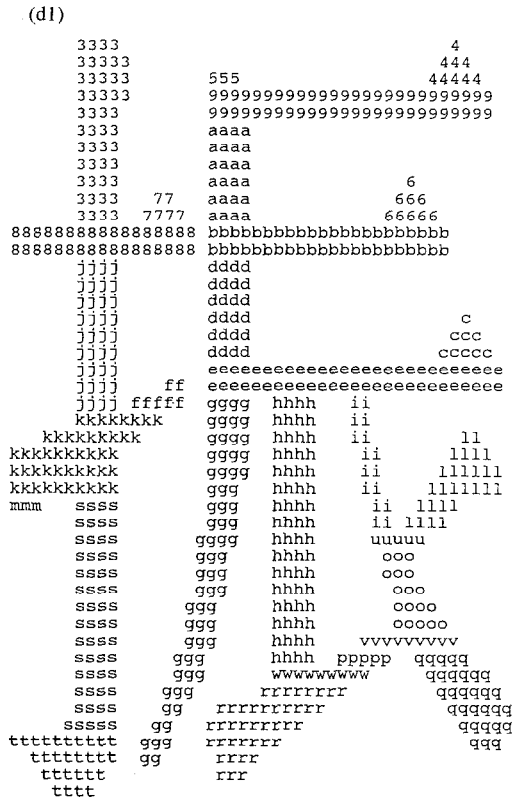
(c1)



(c2)



(d1)



(d2)

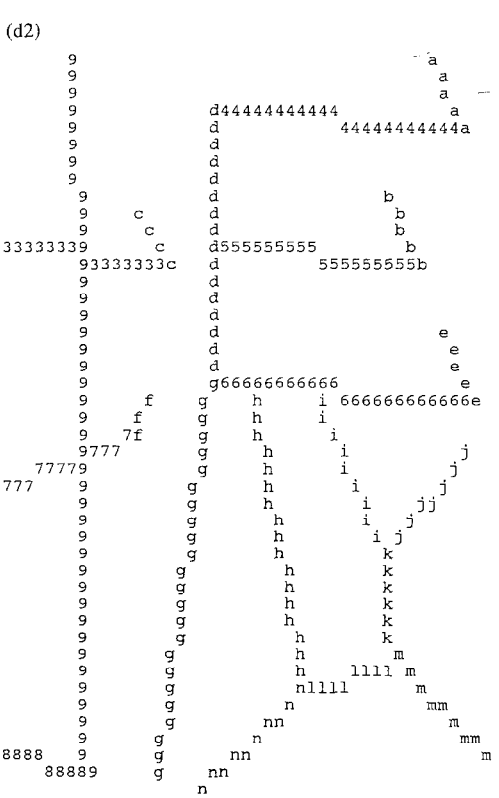


Fig. 11 (*Cont.*)

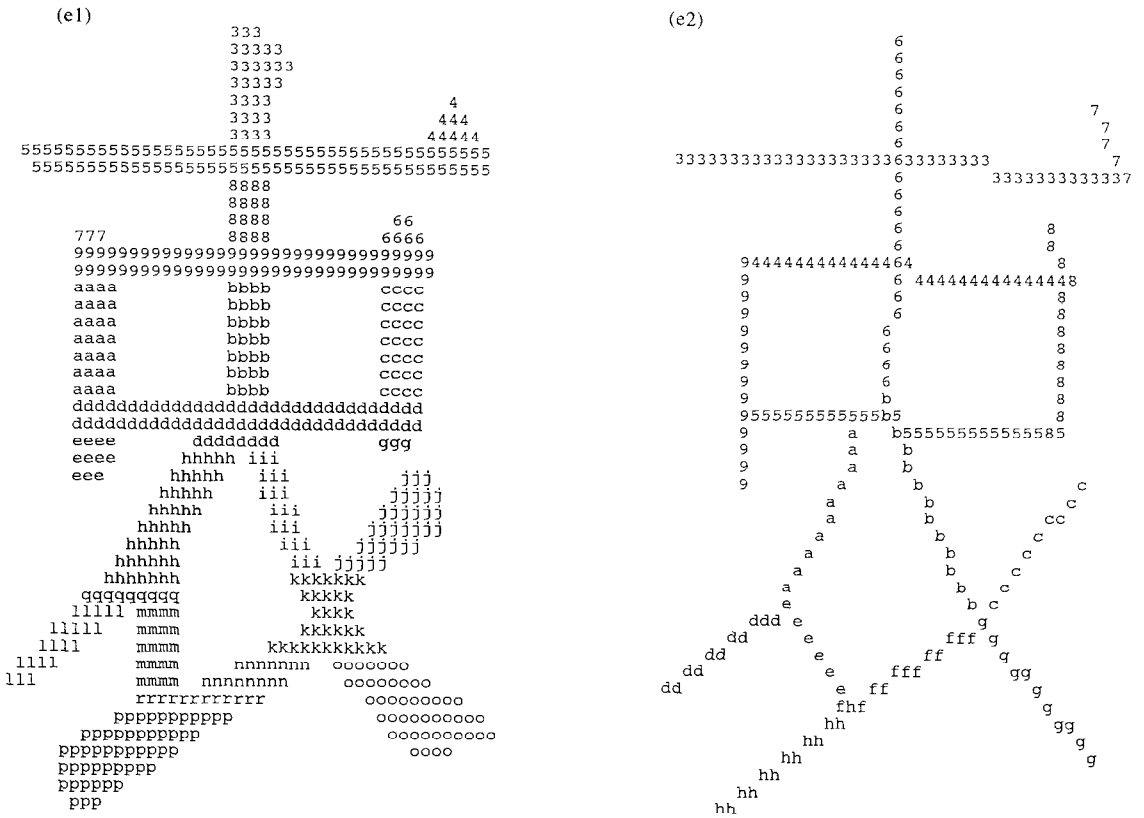


Fig. 11. Stroke pattern "⋊" to be considered. (a) Label 7 is the upward book, label 3 is the crossing stroke and label 2 is the intermediate stroke; (b)–(e) Several variations of the stroke pattern "⋊".

the affected strokes still had to pass through this particular center point. We used line fitting to construct the skeleton, hence the final result was not a one-pixel, four-connected (or eight-connected) graph.

Results for several characters are shown in Fig. 11, 12 and 13. Figure 11 depicts the thinning results of several different variations of the stroke pattern "⋊". The connection between the upward hook and the crossing stroke, called an intermediate stroke [see Fig. 11(a)], varies as follows [in Fig. 11(a)], we do not discuss the strokes labeled 4, 5 and 6 because the characters in Fig. 11(b), (c), (d) and (e) have the same structures]: In Fig. 11(b) the two are connected to become a run (labeled "f" in the figure, with no intermediate stroke). In Fig. 11(c), the top end of the upward hook (labeled "r") and the bottom end of the crossing stroke (labeled "r") are connected directly together (no intermediate stroke). In Fig. 11(d), the upward hook and the crossing stroke are separated by a run (labeled "p" which is an intermediate stroke). In Fig. 11(e), two runs (labeled "n"; which is an intermediate stroke) lie between the upward hook and the crossing stroke.

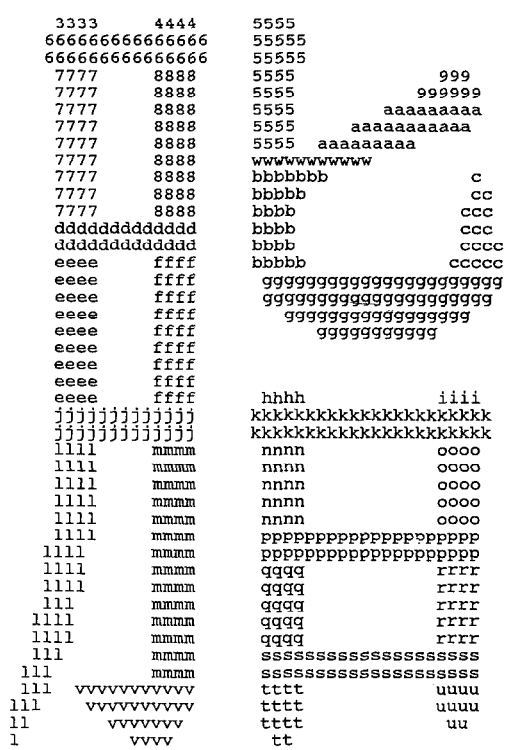
Figure 12 depicts the thinning results for several variations of the stroke pattern "⏃". The "⏃" in Fig. 12(a) is the standard character shape. In Fig. 12(b), the horizontal line on the top of the character (labeled "E" and "F" in the figure) is formed from two horizontal runs, whereas in the second horizontal line a third run appears between the two horizontal runs (labeled "v" and "t"). In Fig. 12(c), there is a horizontal line with two runs (labeled "j"). In the character in Fig. 12(d), the top horizontal line is composed of three runs (labeled "b") and the bottom horizontal line is composed of four runs (labeled "h").

Even though many different types of connections appear in Figs 11 and 12, acceptable thinning results were obtained for all of the characters shown.

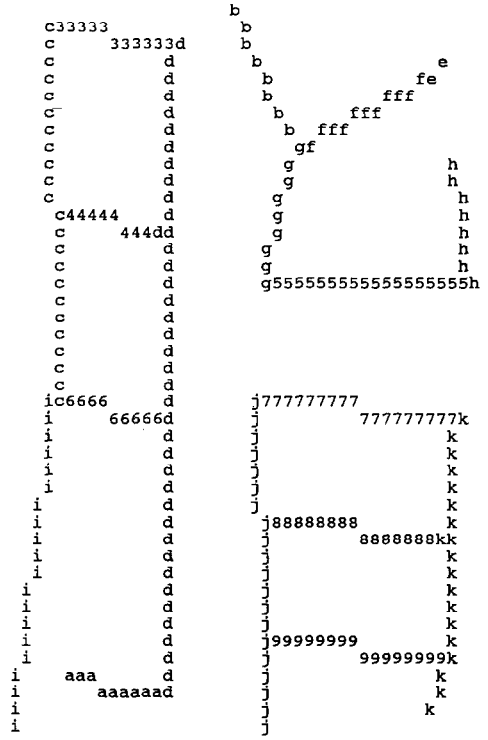
Figure 13 depicts several characters with relatively high stroke counts. A variety of different touching strokes and boundary noise can be seen in these characters. Again, our method achieved reasonably good thinning results for all of the characters shown.

In order to evaluate the thinning results of the proposed method, it is compared with four existing algorithms.[9–12] In Figs 14 and 15 we show the thinning results for the five methods. In general, the proposed method produces better results in two respects: strokes at crossing sections are preserved and the resistance to noise at the character border is high.

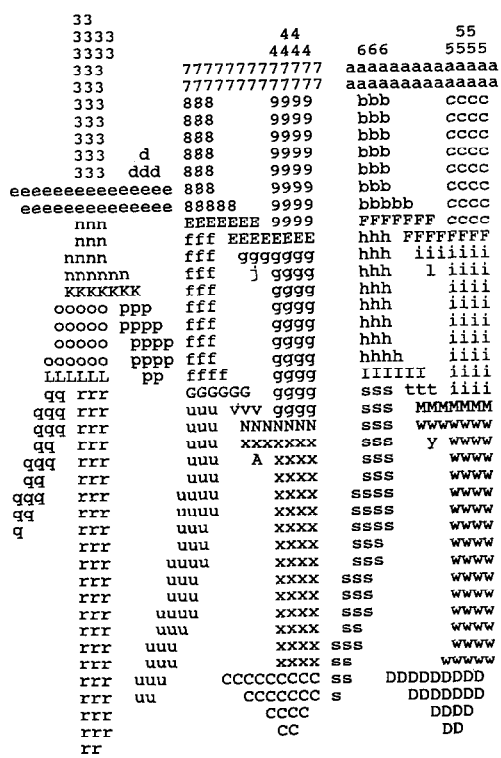
(a1)



(a2)



(b1)



(b2)

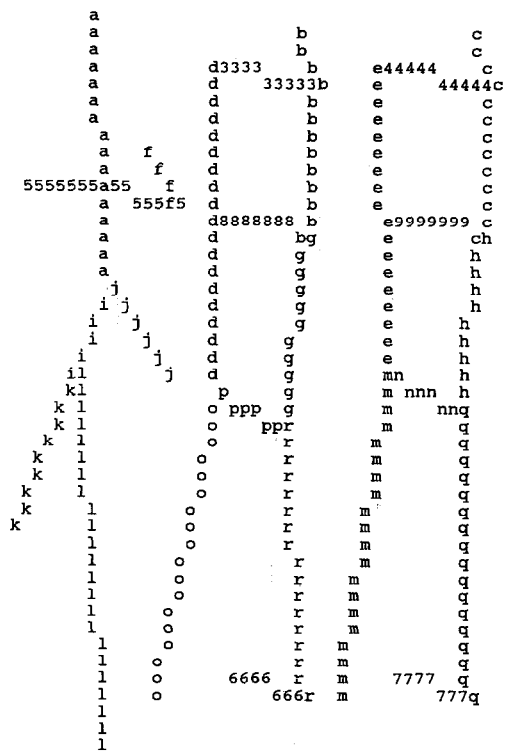


Fig. 12 (*Cont.*)

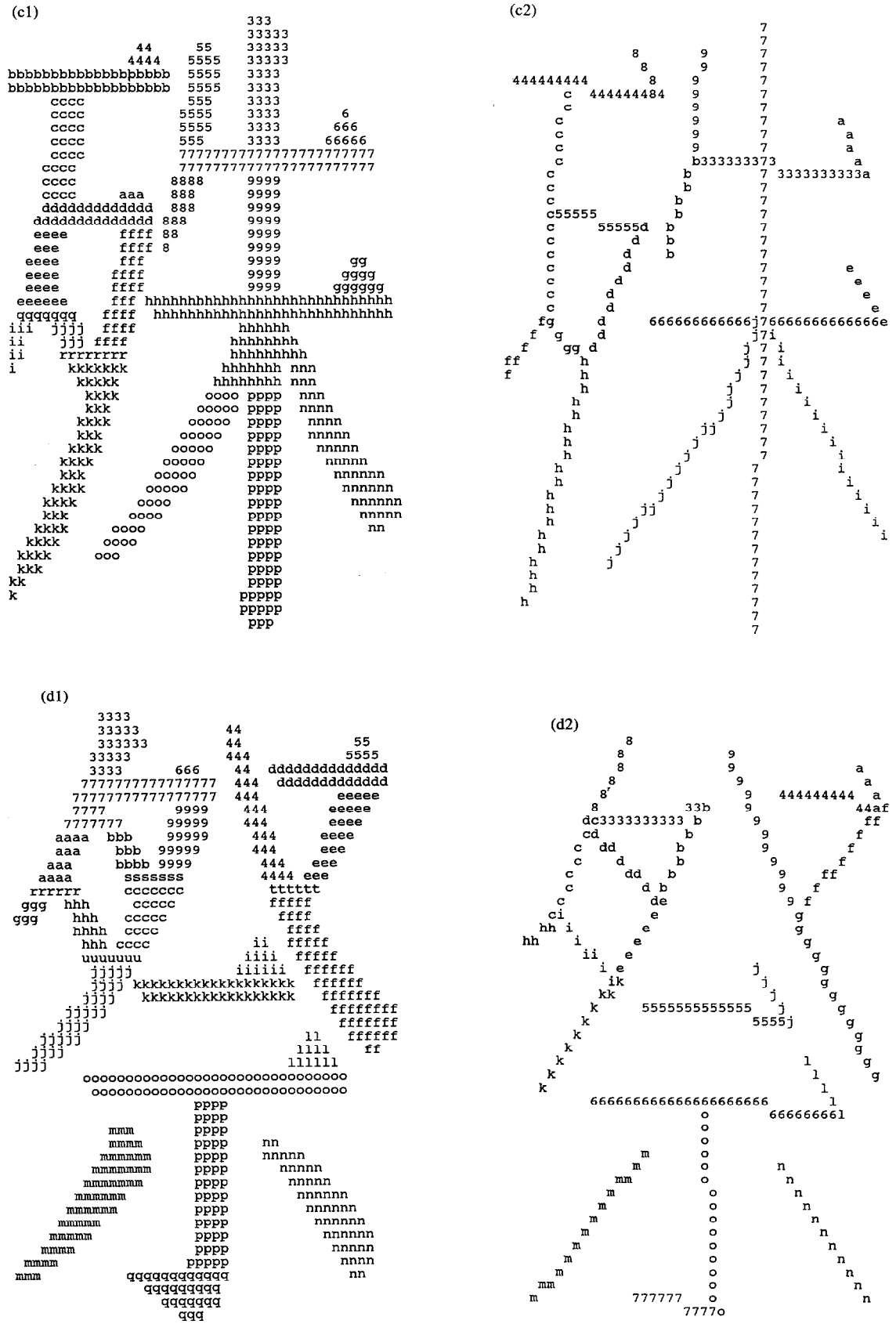
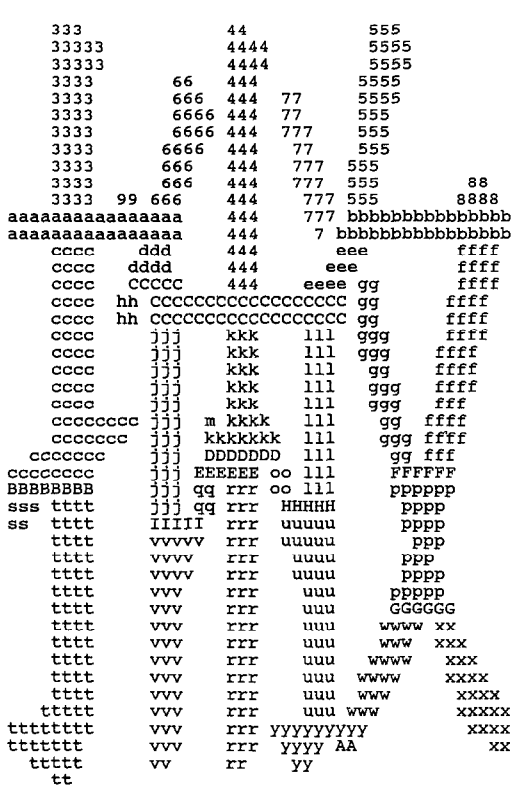
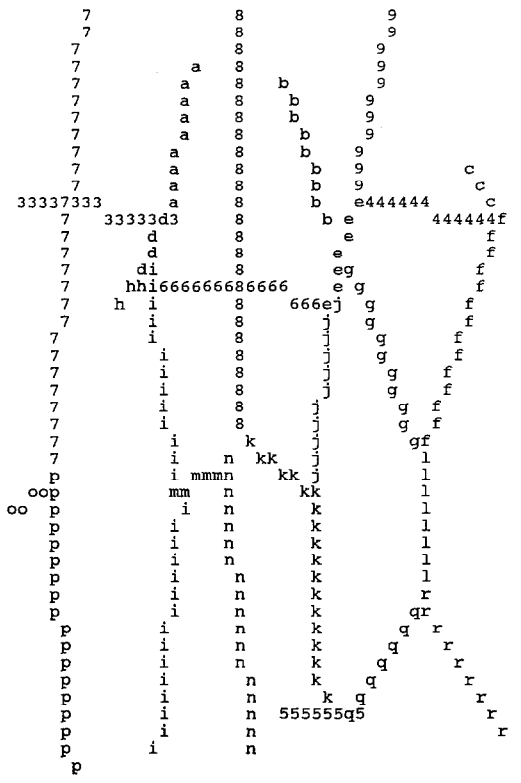


Fig. 12. Several variations of the stroke pattern "亼"

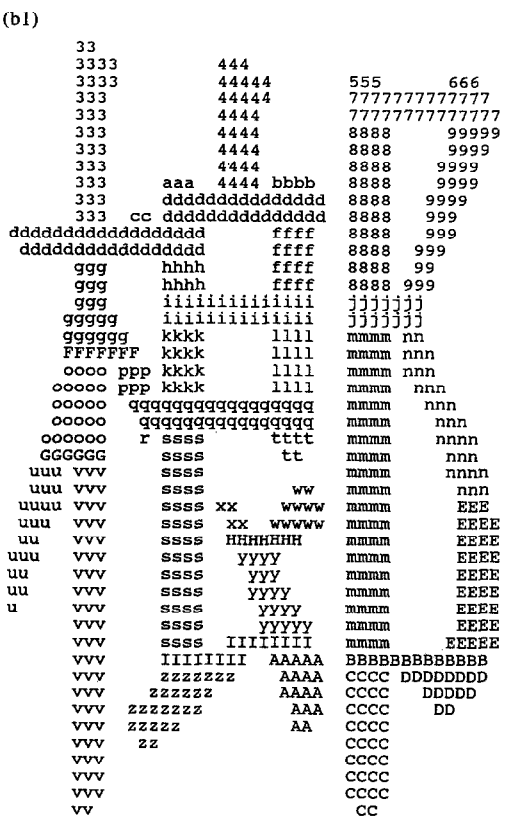
(a1)



(a2)



(b1)



(b2)

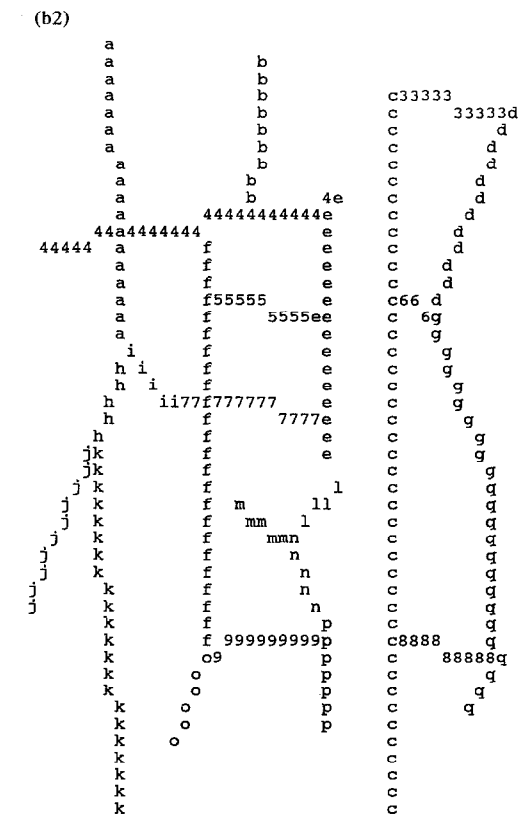


Fig. 13 (*Cont.*)

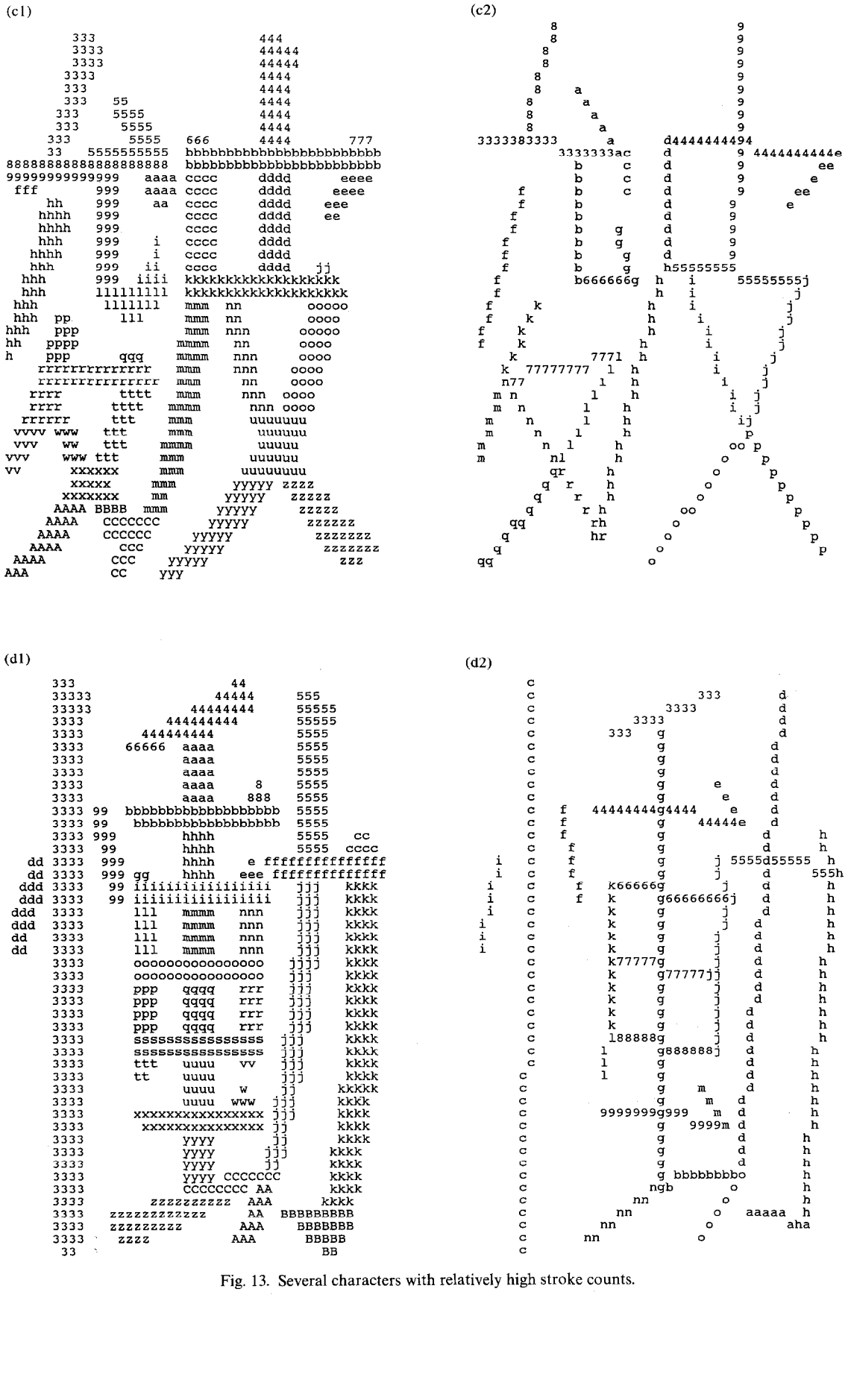


Fig. 13. Several characters with relatively high stroke counts.

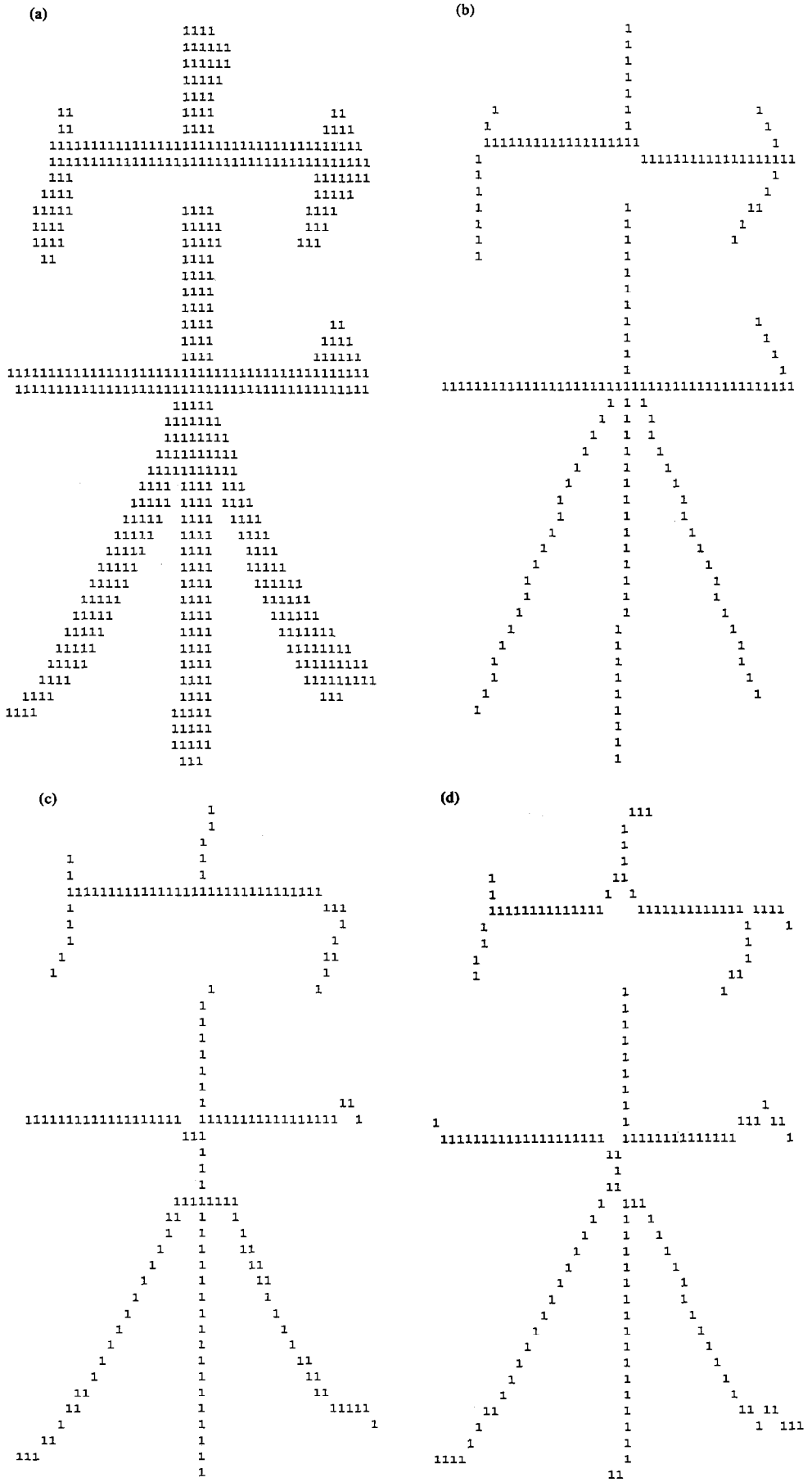


Fig. 14 (*Cont.*)

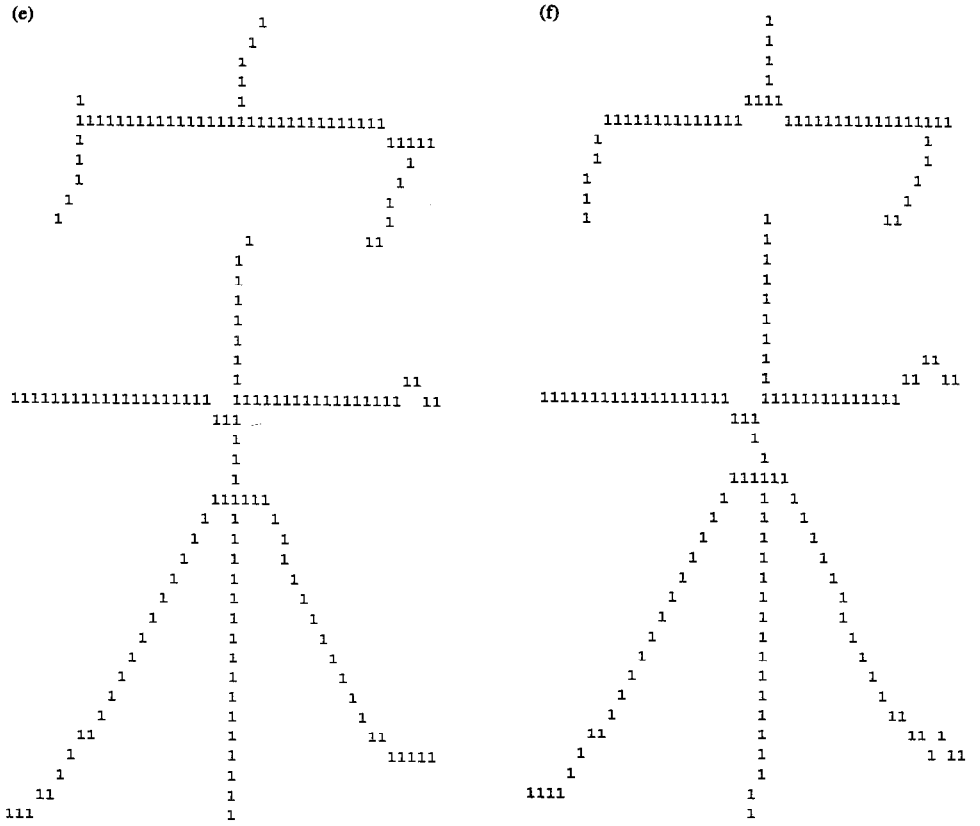


Fig. 14. A sample pattern and its skeletons; (a) an original pattern; (b) ours; (c) C. Y. Suen *et al.*;[9] (d) N. J. Naccache *et al.*;[10] (e) W. H. Tsai *et al.*;[11] (f) C. Y. Suen *et al.*[12]

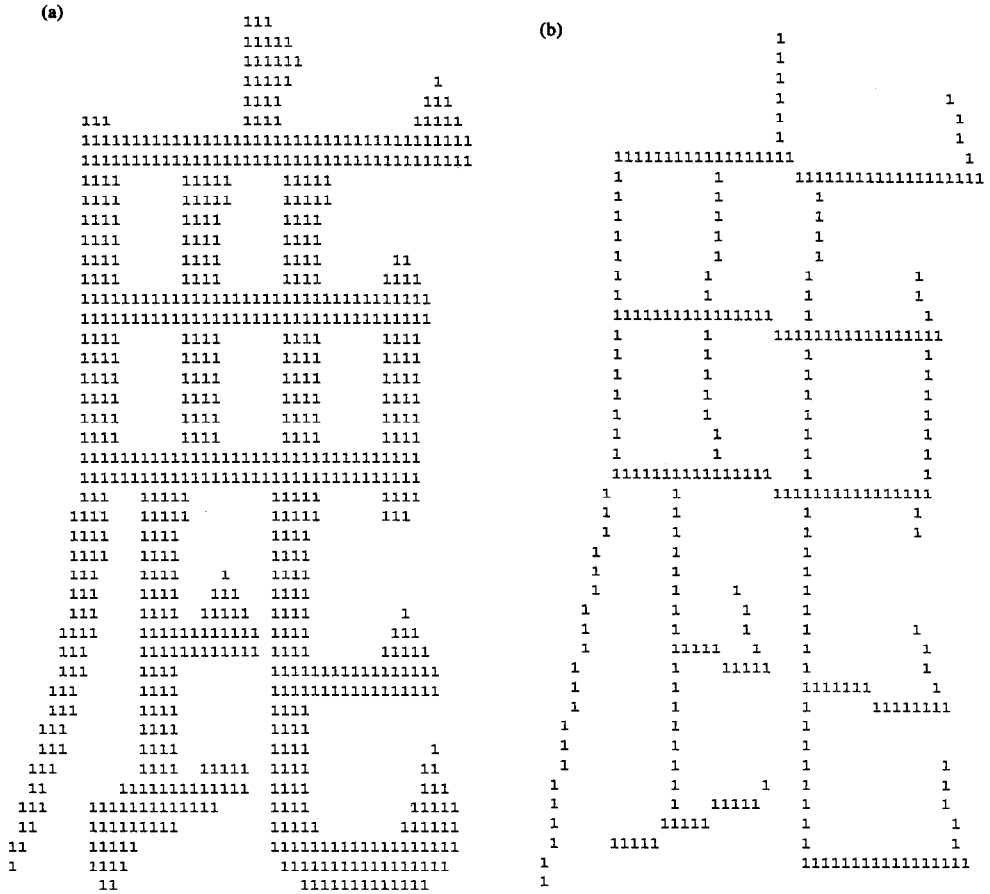


Fig. 15 (*Cont.*)

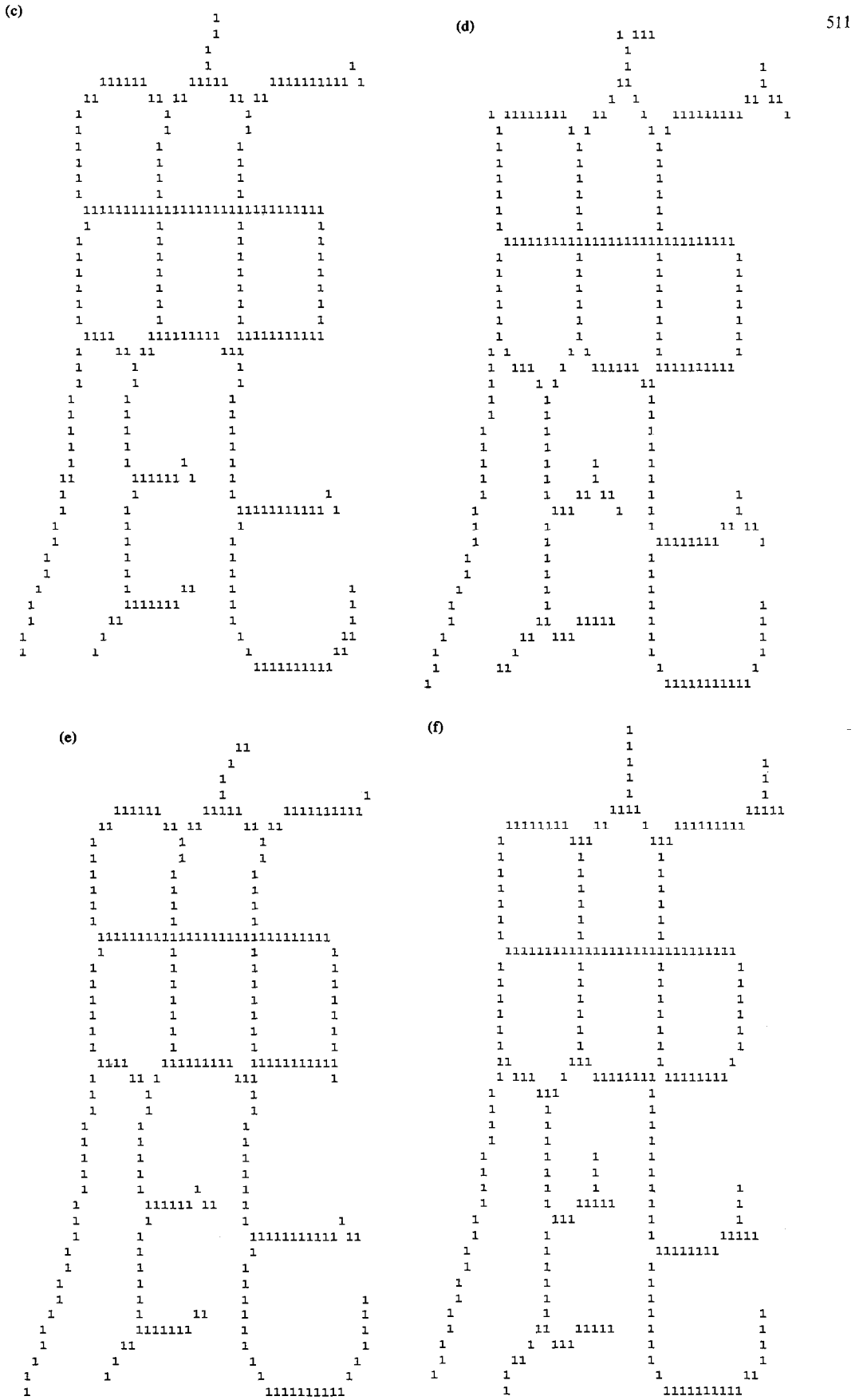


Fig. 15. Another sample pattern and its skeletons; (a) an original pattern; (b) ours; (c) C. Y. Suen *et al.*;[9] (d) N. J. Naccache *et al.*;[10] (c) W. H. Tsai *et al.*;[11] (f) C. Y. Suen *et al.*[12]

7. CONCLUSION

Thinning, or stroke extraction, is the preprocessing step in machine recognition of Chinese characters. A good thinning result can help eliminate many unnecessary decisions in the recognition of Chinese characters, thereby increasing the recognition rate and speed. In this paper, we have proposed an effective thinning method that offers a number of advantages: (1) our method uses runs to perform thinning rather than a window operator which deletes black pixels layer-by-layer, and thus our method provides higher quality thinning results; (2) in identifying the attributes of nodes, intersections between adjacent nodes, and connections between the vertical lines and the left end and righr end of horizontal lines, we eliminate an important potential source of error by examining global features in addition to contour information; (3) noise is detected by comparing the height of a vertical line with half the height of the horizontal line to which the vertical line is connected, instead of merely applying a fixed threshold; (4) to obtain a more reasonable graph, before the thinning stage, we merge the point nodes above and below a horizontal line; (5) in some cases it is difficult to determine whether a node should be identified as a point or a line (knowledge of the structure of Chinese characters is needed). Our policy is that it is better to mistakenly identify a point as a line rather than to identify a line as a point; (6) since our method incorporates a noise-elimination step, it is less likely to generate "hairy" thinning results; (7) last, we use line fitting to construct character skeletons, so if the graph itself is free of errors, the thinning result is likely to be satisfactory.

REFERENCES

1. Louisa Lam, Seong-Whan Lee and Ching Y. Suen, Thinning methodologies—a comprehensive survey, *IEEE Trans. Pattern Analysis Mach. Intell.* **14**(9), 869–885 (1992).
2. Xiaobo Li and Anup Basu, Variable-resolution character thinning, *Pattern Recognition Lett.* **12**, 241–248 (1991).
3. Bei Li and Ching Y. Suen, A knowledge-based thinning algorithm, *Pattern Recognition* **24**(12), 1211–1221 (1991).
4. Paul C. K. Kwok, A thinning algorithm by contour generation, *Comm. ACM* **31**, 1314–1324 (1988).
5. Theo Pavlidis, A vectorizer and feature extractor for document recognition, *CVGIP* **35**, 111–127 (1986).
6. Ling-Hwei Chen, A new approach for handwritten character stroke extraction, *Computer Process. Chinese Oriental Lang.* **6**(1), 1–17 (1992).
7. L. Y. Tseng and C. T. Chuang, An efficient knowledge-based stroke extraction method for multi-font chinese characters, *Pattern Recognition* **12**, 1445–1458 (1992).
8. G. Hu and Z. N. Li, An X-crossing preserving skeletonization algorithm, *Int. J. Pattern Recognition Artif. Intell.* **7**(5), 1031–1053 (1993).
9. T. Y. Zhang and C. Y. Suen, A fast parallel algorithm for thinning digital patterns, *Commun. ACM* **27**, 236–239 (1984).
10. N. J. Naccache and R. Shinghal, SPTA: a proposed algorithm for thinning binary patterns, *IEEE Trans. Syst. Man Cybernetics* **14**(3), 409–418 (1984).
11. R. Y. Wu and W. H. Tsai, A new one-pass parallel thinning algorithm for binary images, *Pattern Recognition Lett.* **13**, 715–723 (1992).
12. A. Arumugam, T. Radhakrishnan, C. Y. Suen and P. S. P Wang, A thinning algorithm based on the force between charged particles, *Int. J. Pattern Recognition Artif. Intell.* **7**(5), 987–1008 (1993).

About the Author—JENN-YIH LIN was born on 3 December 1959 in Taiwan, Republic of China. He received the B.S. degree in control engineering in 1981 and M.S. degree in computer engineering in 1986, both from National Chiao Tung University, Taiwan. In 1983–1984, he worked in Mechanical Industry Research Laboratories, Industrial Technology Research Institute (MIRL, ITRI) at Hsinchu as a software engineer. From 1986 to 1990, he worked as a research assistant in the Chung-Shan Institute of Science and Technology. In 1991, he entered the Institute of computer science and information engineering at National Chiao Tung University, where he is now a Ph.D. candidate. Currently, he is also an instructor at Ming-Hsin Institute of Technology and Commerce. His current research interests include image processing, computer vision, optical character recognition and parallel computation.

About the Author—ZEN CHEN received the B.Sc. degree from National Taiwan University, Taiwan, Republic of China in 1967, the M.Sc. degree from Duke University, Durham, North Carolina, in 1970, and the Ph.D. degree from Purdue University, West Lafayette, Indiana, in 1973, all in electrical engineering. After graduating from Purdue University, he joined Burroughs Corporation, Detroit, Michigan, where he was engaged in the development of a document recognition system. In 1974, he began to teach at National Chiao Tung University, Taiwan, Republic of China. He served as the director of the Institute of Computer Engineering from 1975 to 1980. He spent the academic year 1981–1982 at Lawrence Berkeley Laboratory, University of California, Berkeley, California, as a visiting scientist. Later, in August 1989 he spent about six months at Computer Vision Laboratory of the Center for Automation Research, University of Maryland, College Park, Maryland, as a visiting professor. His current research interests include computer vision, CAD/CAM system, expert system, and parallel algorithms and architectures. Dr Chen is a member of Sigma Xi and Phi Kappa Phi. He is also a member of China Computer Society and Chinese Institute of Electrical Engineering.