MVA'94 IAPR Workshop on Machine Vision Applications Dec. 13-15, 1994, Kawasaki

Stroke Extraction as a Basis for Structural Analysis of Document Images by Modified MCR Expression

Supoj CHINVEERAPHAN, AbdelMalek ZIDOURI, *and* Makoto SATO
Precision and Intelligence Laboratory

TOKYO INSTITUTE OF TECHNOLOGY
4259 Nagatsuta-cho Yokohama-shi Japan 227

Abstract

Since stroke components are the most basic structural features of patterns like characters or lines in images. From a view point of being applicable to document image understanding, it is necessary to provide a good representation that can express such features, so that an interpretation of images can be facilitated. This article presents a new technique used for expressing binary document images. The proposed technique developed from the Minimum Covering Run (MCR) expression to extract stroke components of patterns in images accurately.

1 Introduction

As a first step to develop a system to analyze or recognize patterns contained in images efficiently, it is important to provide a good base representation that can facilitate the interpretation of such information. Since structural features of basic patterns in document images such as characters or tables are horizontal and vertical stroke components, we propose new expression of document image based on the Minimum Covering Run (MCR) expression that can express well such features of text and tabular components of an image.

In the previous works [1]-[2], the MCR expression derived from run representation has been proposed to express binary images by a minimum number of horizontal and vertical runs. It was shown that the MCR method should represent stroke components of characters in a natural way since representation of any stroke should require a minimum number of runs. However, in the actual situation the expression cannot perform accurately for all possible patterns in images. Therefore, we develop a Modified MCR expression to solve this problem.

To utilize the resulted expression for document understanding, strokes are first partitioned into non-overlapping and crossing parts. Then, a description of such information is constructed. The validity of the expression to stroke extraction as well as document image interpretation is shown here by experimental results.

2 Modified MCR Expression

The MCR expression expresses binary document images by a minimum number of runs, called "covering run", both in the horizontal and vertical directions rather than being expressed by either the horizontal or vertical run representation. Since no runs from the same direction cross each other and every black pixel can be considered as a crossing point of one horizontal run and one vertical run, we have defined both run types of images as the partite sets of a bipartite graph. Using the correspondence between binary images and bipartite graphs, the MCR expression can be found by constructing a minimum covering or maximum matching in the corresponding graph. As an implementation, to avoid an expensive processing time caused by the matching algorithm, we have provided Partial Segment Analysis (PSA) [2] as a preprocessing to define covering runs representing elongated components of patterns in an image beforehand, whereas the remaining part of the image is processed further by matching algorithm.

As characters and tables in document images have basic structures of horizontal and vertical strokes, the MCR method should give an insight on stroke representation of such patterns in a natural way– a horizontal stroke is usually represented by a set of adjoining horizontal covering runs while a vertical stroke is expressed by vertical covering runs. However, since the MCR expression does not take into consideration what would be precisely regarded as stroke components, in the actual situation, the expression may fail to extract strokes in cases of

1) noisy patterns: there are many irregularities on the pattern boundaries,

2) patterns containing short strokes: a stroke intersects other ones such that non-overlapping sections of the stroke has its length shorter than its width.

To solve the problem, we develop a new expression

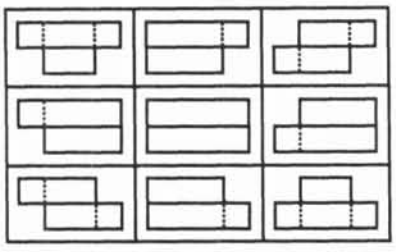

Figure 1: Admissible connecting patterns for two adjoining runs

named Modified MCR expression to extract or partition strokes of patterns more accurately, and also, to reduce the calculating time to obtain the expression.

2.1 Modified MCR Algorithm

A Modified Minimum Covering Run (Modified MCR) expression consists of the main following procedures:

(1) Registration of horizontal and vertical runs:

A raster scan of an input image is carried out, and representing horizontal and vertical runs are registered.

(2) Local Stroke Analysis (LSA):

By leaving the minimal condition for a number of covering runs, this procedure developed from the Partial Segment Analysis of the MCR expression is provided to search for parts of patterns which have *a stroke characteristic* in both horizontal and vertical directions, and appoint covering runs representing them properly. This is described in detail in the next subsection.

(3) Maximum Matching Analysis (MMA):

The remaining parts of the image which have no stroke characteristic will be processed by creating an adjacency matrix for each connected component in the image and constructing a maximum matching in the matrices that gives sets of remaining covering runs. For more detail, see [1].

2.2 Local Stroke Analysis (LSA)

In order to improve an ability to extract strokes, it is necessary to define first what are the essential characteristics of strokes. Based on intuitive observation, the decision criteria for strokes can be formalized as follows:

(a) **Runs Forming Stroke Condition:**

If a segment of binary pattern is formed by a group of adjoining runs where every two adjacent horizontal runs satisfy 9–connecting patterns of Fig.1, the segment is considered as a candidate of non–overlapping section of stroke lying on the vertical direction. This is evident because segments containing only the connecting patterns of Fig.1 will not change their orientations to the horizontal direction; for example, see a segment $ABCD$ in Fig.2(a). This condition is sufficient but not necessary since vertical segments in general possibly contain another type of connecting pattern as shown by dark shaded area in Fig.2(b). However, such another connecting type will not always form only vertical strokes. It may exist in a part of segment along the horizontal direction as well as in a region where strokes intersect as shown by dark shade in Fig.2(c).

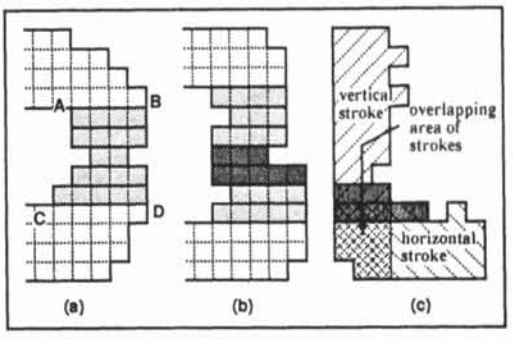

Figure 2: Strokes and their connecting patterns

(b) **Length-Width Condition:**

The most common and efficient way to determine candidate segments mentioned above are stroke or not, is to use the length–versus–width judgement; i.e., a stroke should have its length longer than or equal to its width. In order to figure out a shape of a vertical stroke in Fig.2(a), overlapping areas where the stroke gets contact with other ones should be included in a legitimate component. As depicted in Fig.3(a), the whole stroke being considered in this case is a segment $A'B'C'D'$. We define length and width of the segment $A'B'C'D'$ respectively as $Length = \frac{TArea}{q}$ and $Width = \frac{SArea}{n}$ where,

$$\begin{aligned} SArea &= \text{Area of non–overlapping segment,} \\ TArea &= \text{Area of the stroke being considered,} \\ n &= \text{No. of horizontal runs in segment,} \\ q &= \text{Maximum width of segment.} \end{aligned}$$

Therefore, the segment $A'B'C'D'$ is regarded as a vertical stroke if $Length \geq Width$, or $\frac{TArea}{q} \geq \frac{SArea}{n}$.

From the above idea, we construct a Local Stroke Analysis (LSA) to determine covering runs that represent vertical and horizontal stroke components as:

- For any segment consisting of n adjoining horizontal runs, if connecting patterns of two horizontal runs are within the 9–admissible patterns of Fig.1, and if $\frac{TArea}{q} \geq \frac{SArea}{n}$, the segment is regarded as a non–overlapping section of a vertical stroke and is properly represented by its vertical runs.

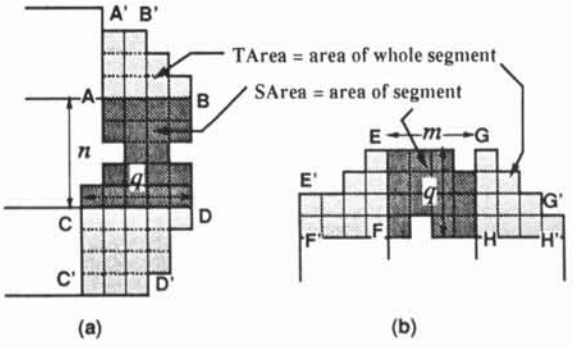

Figure 3: Local Stroke Analysis

- For any segment consisting of m adjoining vertical runs, if connecting patterns of two vertical runs are within the 9-admissible patterns, rotated ones of Fig.1 by 90 degrees, and if $\frac{TArea}{q} \geq \frac{SArea}{m}$, the segment is regarded as a non-overlapping section of a horizontal stroke and is represented by its horizontal runs.

3 A Description of Extracted Strokes and Its Application

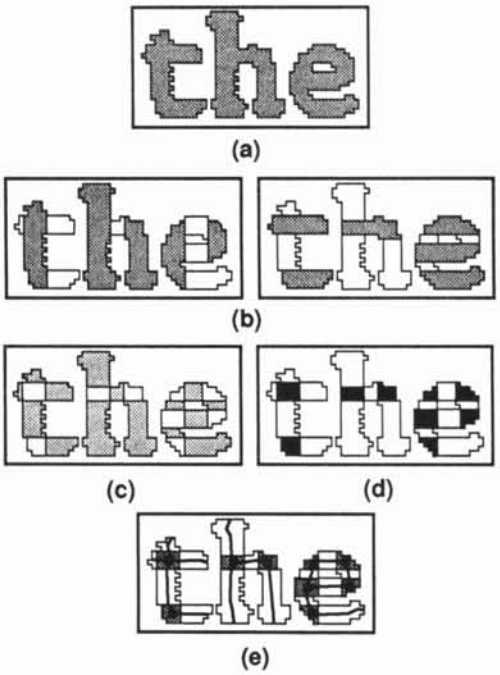

Figure 4: An example of stroke representation
(a) A word "applications", (b) Its MCR expression,
(c) Non-overlapping part of strokes,
(d) Crossing part of strokes, (e) Thinning result

Fig.4 illustrates an example of the Modified MCR expression to show the validity of the proposed technique to stroke extraction. In Fig.4(a), and (b), an image of a word "the" and its Modified MCR are shown respectively. Next, segmented strokes are further decomposed into non-overlapping and crossing parts, as shown in Fig.4(c), and (d), and every part are then labeled. Finally, a description of each (non-overlapping or crossing) part containing the following information is constructed:

Non-overlapping parts: starting run, its run connecting sequence, crossing parts which that part gets contact to.

Crossing parts: center position, non-overlapping parts which that part gets contact to.

To utilize this result in document image analysis, the description may be replaced by expressing non-overlapping strokes by a graph-like approximation as shown in Fig.4(e). Fig.5(b) shows a concrete result of the description by expressing non-overlapping strokes with their center lines extended to crossing points of the resulted expression of an image in Fig.5(a). It shows that the description lays down a basis for structural matching of characters and vectorization of line drawings in the images.

4 Conclusions

A choice of document image representation plays an important role for various types of image processing. In this article, we presented a new technique used for expressing binary document image that can represent well the stroke components of patterns in image. Based on an appropriate description of resulted expression, many applications to document image analysis such as structural matching or vectorization can be easily performed.

References

[1] Douniwa, K., Chinveeraphan, S. and Sato, M., "MCR Expression of Document Images Based on Maximum Matching of Bipartite Graph". *11th IAPR Int. Conf. Pattern Recognition* (Hague, Netherland), vol.3, pp.117-121, Aug.30-Sep.3 1992.

[2] Chinveeraphan, S., Zidouri, A.B.C., and Sato, M., "Fast Algorithms for Minimum Covering Run Expression", *IEICE Trans. Inf. & Syst.*, vol.E77-D, no.3, pp.317-325, Mar. 1993.

[3] Boatto, L., Consorti, V., Del Buono, V., Di Zenzo, S., Eramo, V., Esposito, A., Melcarne, F., Meucci, M., Morelli, A., Mosciatii, M., Scarci, S. and Tucci, M., "An Interpretation System for Land Register Maps", *Computer*, no.7, pp.25-33, 1992.

Figure 5:

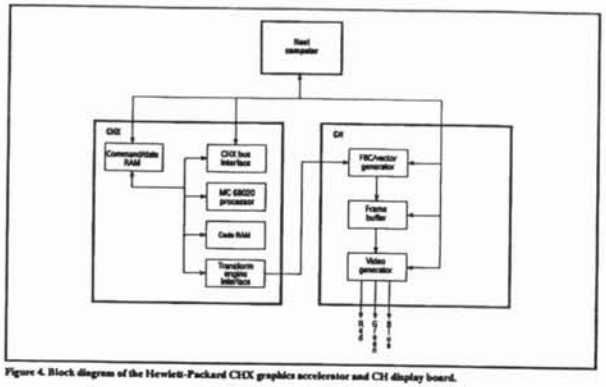

Figure 4. Block diagram of the Hewlett-Packard CHX graphics accelerator and CH display board.

as a buffer for sending graphics commands and data from the host computer to the accelerator; code RAM containing the code and data for the MC68020 processor; and various bus interfaces.

To use the HP CHX, the host computer writes graphics commands to the command/data RAM. The MC68020 reads those commands, applies appropriate transformations, scan-converts polygons, and sends commands to the display board to draw vectors or fill rectangular areas. The command/data RAM has two ports so that the host can write commands to it at the same time that the MC68020 reads commands from it.

The accelerator keeps 32 *graphics contexts* in the code RAM. A graphics context on the HP CHX accelerator consists of the current set of drawing attributes, drawing colors, and other hardware state information. There is usually a one-to-one mapping between graphics contexts, windows on the screen, and graphics applications. (However, an application can access a single window via multiple graphics contexts when it needs more than one graphics contest for drawing to one window.) As Figure 5 shows, each graphics context on the accelerator identifies the window it is associated with using a window identification field. The clip list index table has an entry for each window. Each entry includes a window identifier, the number of clipping rectangles for the window, and a pointer to the first clipping rectangle.

The window server process directly writes the clip list index table and the clip list. Whenever the window configuration changes, the window server recomputes the clip lists for all windows, waits for the accelerator to finish processing the current command/data buffer, halts the MC68020 processor, writes the clipping information to the code RAM, then restarts the MC68020. Figure 6 gives an example of data structures that the window server downloads to the accelerator.

The rendering code that runs on the MC68020 draws a graphics primitive once for each rectangle in the window clip list, setting the hardware clipping limits on the accelerator to the limits of each rectangle before drawing the primitive:

```
for each rectangle in the window clip list do
    set the clip limits on the accelerator to the limits of the
        current rectangle
    draw the primitive
```

Implementation of the above algorithm on the HP CHX is optimized to reject primitives that lie completely outside the clip rectangle. The time to render a primitive increases linearly as the number of rectangles making up a window increases.

The size of the HP CHX accelerator's RAM limits space available to represent the window clipping rectangles to approximately 16 Kbytes. The accelerator represents rectangles

(a)

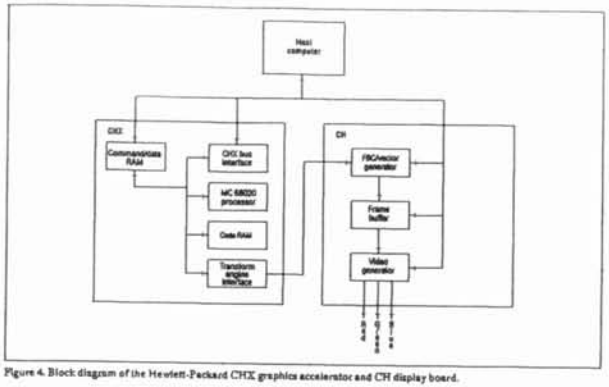

Figure 4. Block diagram of the Hewlett-Packard CHX graphics accelerator and CH display board.

as a buffer for sending graphics commands and data from the host computer to the accelerator; code RAM containing the code and data for the MC68020 processor; and various bus interfaces.

To use the HP CHX, the host computer writes graphics commands to the command/data RAM. The MC68020 reads those commands, applies appropriate transformations, scan-converts polygons, and sends commands to the display board to draw vectors or fill rectangular areas. The command/data RAM has two ports so that the host can write commands to it at the same time that the MC68020 reads commands from it.

The accelerator keeps 32 *graphics contexts* in the code RAM. A graphics context on the HP CHX accelerator consists of the current set of drawing attributes, drawing colors, and other hardware state information. There is usually a one-to-one mapping between graphics contexts, windows on the screen, and graphics applications. (However, an application can access a single window via multiple graphics contexts when it needs more than one graphics context for drawing to one window.) As Figure 5 shows, each graphics context on the accelerator identifies the window it is associated with using a window identification field. The clip list index table has an entry for each window. Each entry includes a window identifier, the number of clipping rectangles for the window, and a pointer to the first clipping rectangle.

The window server process directly writes the clip list index table and the clip list. Whenever the window configuration changes, the window server recomputes the clip lists for all windows, waits for the accelerator to finish processing the current command/data buffer, halts the MC68020 processor, writes the clipping information to the code RAM, then restarts the MC68020. Figure 6 gives an example of data structures that the window server downloads to the accelerator.

The rendering code that runs on the MC68020 draws a graphics primitive once for each rectangle in the window clip list, setting the hardware clipping limits on the accelerator to the limits of each rectangle before drawing the primitive:

```
for each rectangle in the window clip list do
    set the clip limits on the accelerator to the limits of the
        current rectangle
    draw the primitive
```

Implementation of the above algorithm on the HP CHX is optimized to reject primitives that lie completely outside the clip rectangle. The time to render a primitive increases linearly as the number of rectangles making up a window increases.

The size of the HP CHX accelerator's RAM limits space available to represent the window clipping rectangles to approximately 16 Kbytes. The accelerator represents rectangles

(b)