A Modified Parallel Thinning Algorithm

Y. Y. Zhang and P. S. P. Wang

College of Computer Science
Northeastern University
Boston, MA 02115

Abstract

A parallel thinning algorithm[1](Holt etc.) is compared with algorithm[2] (Rutovitz) and [3](Zhang and Suen). Analyses and experiments show that the algorithm[1] is similar to the algorithm[2]. A heuristic modification to algorithm[2] is also proposed and the modified algorithm is faster than algorithm[1].

1 Introduction

In image processing and pattern recognition, a binary digitized pattern can be represented by a matrix, where each element is either 1 (dark point) or 0 (white point) and these points are called pixels. It is here assumed that the patterns consist of these elements which have value 1. 'Thinning' is a process that deletes the dark points and transforms the pattern into 'thin' line drawing known as a skeleton. In parallel processing, the value of a pixel at the n-th iteration depends on the values of the pixel and its neighbors at the (n-1)-th iteration. Thus all the pixels of the pattern can be processed simultaneously. Parallel algorithms for obtaining the skeleton of a pattern have been described by other authors [1-5]. Holt etc. recently presented an improved parallel thinning algorithm[HS][1]. They said that the time is improved over 40 percent on "fast parallel thinning algorithm by Zhang and Suen" [ZS][3]. After our investigations algorithm HS is similar to algorithm RD proposed by Rotovitz[2]. In the section 3, we propose a modified algorithm which gives some empirical results comparing algorithm HS, ZS and RD.

2 Comparison of HS and RD

In algorithm RD, it is assumed that a 3x3 window is used for each picture element. That is, the values of the eight neighbors of a central element(p1) are used in the calculation of its values for the next iteration. The neighboring values can be denoted by the eight directions (p2, p3, ..., p9) shown in Figure 1.

```
p9 p2 p3
p8 p1 p4
p7 p6 p5
```

Figure 1. Pixel p1 and its neighbors in algorithm RD

In algorithm RD, the algorithm was described as follows: Let A(p1) be the number of pattern 01 in the ordered set p2, p3, ..., p8, p9, p2 of the neighbors of p1. Let B(p2) be the number of nonzero neighbors of p1. Then a point p1 is deleted from the figure if all of the following 5 tests (a0-0)-(a4-0) return the value TRUE:

(a0-0) p1=1
(a1-0) 2<=B(p1)<=6
(a2-0) A(p1)=1
(a3-0) p2p4p8p0 or A(p2)<=1
(a4-0) p2p4p6p0 or A(p4)<=1

This algorithm yields connected skeletons which are not sensitive to contour noise.

In algorithm CH, a digital picture is represented by a logical matrix of true and false elements(is and 0s). The neighboring values are denoted by the compass directions(north, south west, etc.) shown in Fig.2.

```
NW N NE
W C E
SW S SE
```

Figure 2. Pixel p1 and its neighbors in algorithm HS

A pixel survives an iteration if the conditions represented by the expression(except 2x2 square)

\[
vC \quad \text{and} \quad (\text{edge}_C \quad \text{or} \quad (\text{edge}_E \quad \text{and} \quad vW \quad \text{and} \quad vS) \\
\text{or} \quad (\text{edge}_N \quad \text{and} \quad vW \quad \text{and} \quad vE))
\]

(1)

is true. Where ~ denotes 'not' and v denotes value at a location. vC means the value of C is true, ~edgeC means C is not on the edge. edgeE and vN and vS is true' means that its east neighbor is on an edge and it is not on a corner. In this case it maintains the westward bias of the original pattern for vertical stroke of width 2. Similarly, the south edge of a horizontal 2-strok is reserved if the expression "edgeN and vW and vE" is true. These conditions ensure that connectedness is preserved[1]. For removal, the expression can be represented as follows:

\[
vC \quad \text{and} \quad \text{edge}_C \quad \text{and} \quad (\text{edge}_E \quad \text{or} \quad vW \quad \text{or} \quad vS) \\
\text{and} \quad (\text{edge}_N \quad \text{or} \quad vW \quad \text{or} \quad vE)
\]

(2)
If we use the notation in (a-0) to represent boolean procedure edgeC (see Appendix A). The edgeC is equal to: 
\( p1 \) and \( 2 \leq B(p1) \leq 6 \) and \( A(p1) := 1 \). 

Therefore, the thinning condition (2) for algorithm CH using the notation of algorithm RD is:

\[
\begin{align*}
(b0-0) & \ p1 \neq 1 \\
(b1-0) & \ 2 \leq B(p1) \leq 6 \\
(b2-0) & \ A(p1) = 1 \\
(b3-0) & \ p2 \neq p4 \neq p8 \neq 0 \ or \ A(p2) \neq 1 \\ & \ or \ not \ (2 \leq B(p2) \leq 6) \\
(b4-0) & \ p2 \ neq p4 \neq p6 \neq 0 \ or \ A(p4) \neq 1 \\ & \ or \ not \ (2 \leq B(p4) \leq 6)
\end{align*}
\]

Experiment shows that algorithms HS is similar to algorithm RD.

3 Modified Thinning Algorithm

In this paper we propose a modified algorithm (MA), which is faster than the algorithm HS and RD. For pixel \( p1 \) we use the 4*4 neighbors (see Fig.3).

![Figure 3. Pixel p1 and its neighbors in algorithm MA](image)

A point \( p1 \) is deleted from the pattern if all of the following five conditions are satisfied.

\[
\begin{align*}
(c0-0) & \ p1 \neq 1 \\
(c1-0) & \ 2 \leq B(p1) \leq 6 \\
(c2-0) & \ A(p1) = 1 \\
(c3-0) & \ p2 \neq p4 \neq p8 \neq 0 \ or \ p11 \neq 1 \\
(c4-0) & \ p2 \neq p4 \neq p6 \neq 0 \ or \ p15 \neq 1
\end{align*}
\]

Why can we simply use \( p11 = 1 \) and \( p15 = 1 \) in Algorithm MA instead of \( A(p2) \neq 1 \) and \( A(p4) \neq 1 \) in Algorithm RD?

(i). In Algorithm RD, if \( p2 = 1 \) and \( p4 = 1 \) and \( p6 = 1 \) and \( p8 = 1 \) then this pixel can never be deleted. Since conditions \( (c1-0) \) and \( (c2-0) \), i.e., \( 2 \leq B(p1) \leq 6 \) and \( A(p1) = 1 \),
can not be TRUE.

(ii). For \( p2 = 1 \) and \( p4 = 1 \) and \( p8 = 1 \), a pixel is deleted from the digital pattern if all of the following conditions (a-1) are satisfied.

\[
\begin{align*}
(a0-1) & \ p1 \neq 1 \\
(a1-1) & \ 2 \leq B(p1) \leq 6 \\
(a2-1) & \ A(p1) = 1 \\
(a3-1) & \ A(p2) = 1 \\
(a4-1) & \ p6 \neq 0
\end{align*}
\]

(iii). For \( p2 = 1 \) and \( p4 = 1 \) and \( p6 = 1 \) a pixel can be deleted only if all of the following conditions (a-2) are satisfied.

\[
\begin{align*}
(a0-2) & \ p1 \neq 1 \\
(a1-2) & \ 2 \leq B(p1) \leq 6 \\
(a2-2) & \ A(p1) = 1 \\
(a3-2) & \ p8 \neq 0 \\
(a4-2) & \ A(p4) = 1
\end{align*}
\]

In (ii), condition (a-1) is equal to:

\[
\begin{align*}
0 & \ 1 \ 0 \\
1 & \ 1 \ 1 \\
1 & \ p1 \ 1 \\
* & \ 0 \ *
\end{align*}
\]

\( * \) means one or all of them are ‘0’.

Clearly, if we add the following cases then the connectivity of pattern is still reserved.

\[
\begin{align*}
0 & \ 1 \ 1 \\
1 & \ 1 \ 1 \\
1 & \ p1 \ 1 \\
* & \ 0 \ *
\end{align*}
\]

\( (c) \)

\( (d) \)

\( (a), (b), (c) \) and \( (d) \) is equal to the following (c-1):

\[
\begin{align*}
(c0-1) & \ p1 \neq 1 \\
(c1-1) & \ 2 \leq B(p1) \leq 6 \\
(c2-1) & \ A(p1) = 1 \\
(c3-1) & \ p11 \neq 1 \\
(c4-1) & \ p6 \neq 0
\end{align*}
\]

We can also construct condition (c-2) instead of (a-2)

\[
\begin{align*}
(c0-2) & \ p1 \neq 1 \\
(c1-2) & \ 2 \leq B(p1) \leq 6 \\
(c2-2) & \ A(p1) = 1 \\
(c3-2) & \ p8 \neq 0 \\
(c4-2) & \ p15 \neq 1
\end{align*}
\]

Combining (c-1) and (c-2), we get Algorithm MA.

4 Complexity

If we use the survival notation in HS then the four thinning algorithms above can be rewritten as follows.

Algorithm RD:
\( vV \) and (“edgeC or (SedgeE and vW and vS and vE))
or (SedgeW and vW and vE and vW))

Algorithm ZS:
The first subiteration is:
(1) \( vV \) and
(“edgeC or (vE and vS and (vW or vW)))
The second subiteration is:
(2) \( vV \) and
(“edgeC or (vW and vW and (vE or vS)))
Algorithm HS:

\[ v_C \text{ and } (\neg \text{edgeC} \text{ or } (\text{edgeE} \text{ and } v_W \text{ and } v_S) \text{ or } (\text{edgeN} \text{ and } v_W \text{ and } v_E)) \]

Algorithm MA:

\[ v_C \text{ and } (\neg \text{edgeC} \text{ or } (v_W \text{ and } v_E \text{ and } v_S \text{ and } v_{EE}) \text{ or } (v_N \text{ and } v_E \text{ and } v_W \text{ and } \neg v_{WN})) \]

Where NN=p11, EE=p15, Sedge is shown in Appendix B. It is assumed that performing an assignment or a logical operation is one unit time(u). Execution of an if-then statement is assumed to require one logical operation. The edge function require 85 u, and the Sedge function requires 34 u. The time for four algorithm is shown in Table 1.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>unit of time</th>
</tr>
</thead>
<tbody>
<tr>
<td>RD</td>
<td>173</td>
</tr>
<tr>
<td>ZS</td>
<td>91 to 182</td>
</tr>
<tr>
<td>CH</td>
<td>268</td>
</tr>
<tr>
<td>MA</td>
<td>95</td>
</tr>
</tbody>
</table>

Table 1. The comparison of the complexity

5 Implementation

The comparison of time consumed of Algorithm RD, ZS, HS and MA for the pattern 'hexagon', 'leaf', 'body', 'A' and 'mountain' are shown in Table 2 (the unit is sec.). These algorithms were coded in TURBO PASCAL and run on a IBM Personal Computer.

<table>
<thead>
<tr>
<th>pattern</th>
<th>'hexagon'</th>
<th>'leaf'</th>
<th>'body'</th>
<th>'A'</th>
<th>'mountain'</th>
</tr>
</thead>
<tbody>
<tr>
<td>algo RD</td>
<td>10.49</td>
<td>15.93</td>
<td>3.82</td>
<td>3.18</td>
<td>6.81</td>
</tr>
<tr>
<td>algo ZS</td>
<td>12.19</td>
<td>13.90</td>
<td>6.11</td>
<td>2.47</td>
<td>3.37</td>
</tr>
<tr>
<td>algo CH</td>
<td>12.47</td>
<td>19.39</td>
<td>3.70</td>
<td>3.67</td>
<td>7.33</td>
</tr>
<tr>
<td>algo MA</td>
<td>7.19</td>
<td>10.21</td>
<td>3.85</td>
<td>2.03</td>
<td>4.22</td>
</tr>
</tbody>
</table>

Table 2. The comparison of time consumed

6 Discussions

In this paper, we have presented a modified thinning algorithm MA and compared algorithm HS with RD, ZS and MA. The analyses and experiments show that the algorithm HS is similar to algorithm RD and the modified algorithm MA is faster than RD, ZS and HS.

In paper[1], authors proposed a communication programming technique to save thinning time for whole pattern(not for thinning conditions). Even using this technique the algorithm HS is similar to the algorithm RD.

For preserving connectedness and 2*2 square pattern please see paper[1,4,5,7].

References


Appendix A (from literature[1])

Function edge: Boolean;
Var t00, t01, t01s, t11: Boolean;
Procedure check(v1, v2, w3: Boolean);
Begin
If \(\neg v_2 \text{ and } (\neg v_1 \text{ or } v_3)\) Then t00:=True;
If v2 And (v1 Or v3) Then t11:=True;
If (\(\neg v_1 \text{ And } v_2\) Or (v2 And v3)) Then
Begin t01s:=t01; t01:=true
End;
Begin {edge}
t00:=False; t01:=False;
t01s:=False; t11:=False;
check(vNW, vN, vNE); check(vNE, vE, vSE);
check(vSE, vS, vSW); check(vSW, vW, vNW);
edge:=vC And t00 And t11 And \(\neg t01s\);
End;

Appendix B

Function Sedge: Boolean;
Var tois, t01: Boolean;
Procedure Scheck (v1, v2, w3: Boolean);
Begin
If (\(\neg v_1 \text{ And } v_2\) Or (v2 And v3)) Then
Begin t01s:=t01; t01:=True
End;
Begin
t01s:=True; t01:=False;
Scheck(vNW, vN, vNE); Scheck(vNE, vE, vSE);
Scheck(vSE, vS, vSW); Scheck(vSW, vW, vNW);
Sedge := vC and t01s
End;