A Parallel Thinning Algorithm with Two-Subiteration that Generates One-Pixel-Wide Skeletons

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

College of Computer Science
Northeastern University
Boston, MA 02115

email:pwang@ccs.neu.edu, (617)373-3711, fax(617)373-5121

Abstract

Many algorithms for vectorization by thinning have been devised and applied to a great variety of pictures and drawings for data compression, pattern recognition and raster-to-vector conversion. But parallel thinning algorithms which generate one-pixel-wide skeletons can have difficulty preserving the connectivity of an image. In this paper, we propose a 2-subiteration parallel thinning algorithm with template matching (PTA2T) which preserves image connectivity, produces thinner results, maintains very fast speed, and generates one-pixel-wide skeletons.

1. Introduction

In dealing with pattern recognition, data compression and raster-to-vector conversion, the study of vectorization has attracted more and more attention in industries and academia, since it can be applied to compressing data and retaining significant features of the image. But, vectorization algorithms require one-pixel-wide lines as input [3]. One can not use some published algorithms for vectorizing if there are redundant pixels[1] in skeletons. Therefore, in such cases, one should do some additional work (additional thinning) to remove the redundant pixels.

If you want to jump the additional thinning step you should use the algorithms that generate one-pixel-wide skeletons. In this paper, we will give the definition of one-pixel-wide skeleton and design a 2-subiteration thinning algorithm with templates which generates one-pixel-wide skeletons.

2. Definitions and Notations

The image is the set of all points in the Euclidean plane. A set is finite if it contains only a finite number of elements. Let S be a finite image and B = { 0, 1 }. A finite binary image is a function V mapping S into B. Then the binary image is uniquely defined by its foreground set P (or black pixel set). While those of the complement set will be termed background set (or white pixel set).

$$P = \{ p \mid p \in S \wedge V(p) = 1 \}$$

Definition 1. N(p) is the set of all neighbors of p and is termed the **neighborhood** of p (Figure 1 [W_1]). Neighbors having even numbers (p[0], p[2], p[4], p[6] or north(p), east(p), south(p), west(p)) are 4-**neighbors** of *p*; neighbors with odd numbers (p[1], p[3], p[5], p[7] or north-east(p), south-east(p), south-west(p), north-west(p)) together with the 4-neighbors, are call 8-**neighbors** of *p*.

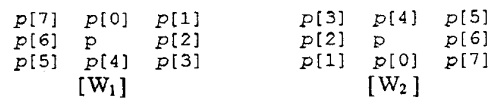

Figure 1. Window W_1 and W_2

Definition 2. **Weight Number** of p, $WN_1(p)$, is defined:

$$WN_1(p) = \sum_{k=0}^{7} p[k] * 2^k$$

For a 3*3 window W_1 the weight numbers are among 0 and 255. In the rest of this paper we always use weight number $WN_1(p)$ indicate point p. For example, pixel 86

1015-4651/96 $5.00 © 1996 IEEE
Proceedings of ICPR '96

means pixel p with the weight number 86. A set that consists of all weight numbers is called WS = { 0, 1, ..., 255 }.

Definition 3. **Neighbor Number** of p, NN(p), is the number of nonzero neighbors of current pixel p:

$$NN(p) = \sum_{k=0}^{7} p[k]$$

Definition 4. A point, p in P, is an **End Point** if NN(p) = 1. All end points consist of a set termed ES.

ES = { 1, 2, 4, 8, 16, 32, 64, 128 }.

Definition 5. **Connection Number** of p, CN(p), is defined:

$$CN(p) = \sum_{k=0,2,4,6} \overline{p[k]} * (p[k+1] \cup p[k+2])$$

Note that $\overline{p[k]} = 1 - p[k]$ and p[8] = p[0].

Definition 6. A point, p in P, is termed **Globally Removable** if CN(p) = 1 and NN(p) > 1. All globally removable points consist of a set termed GRS.

```
GRS={ 3,  5,  6,  7, 12, 13, 14, 15, 20, 21, 22, 23, 24,
  28, 29, 30, 31, 48, 52, 53, 54, 55, 56, 60, 61, 62, 63,
  65, 67, 69, 71, 77, 79, 80, 81, 83, 84, 86, 88, 89, 91,
  92, 94, 96, 97, 99,101,103,109,111,112,113,115,116,118,
 120,121,123,124,126,129,131,133,135,141,143,149,151,157,
 159,181,183,189,191,192,193,195,197,199,205,207,208,209,
 211,212,214,216,217,219,220,222,224,225,227,229,231,237,
 239,240,241,243,244,246,248,249,251,252,254 }  - 108
```

Definition 7. A point, p in P, is termed **irreducible** if p is not a globally removable point. All irreducible points consist of a set termed IPS. IPS = WS - GRS

Definition 8 **Locally Removable Set or Removable Set,** LRS, is a subset of GRS and defined for subiteration. The element in LRS is called *removable point* and the neighbors of p that are in LRS are called *removable neighbors* of p. For subiteration parallel thinning algorithms, we define locally removable sets. For example, in 4-subiteration, we define 4 locally removable sets: LRS4[1] = {p | p in GRS and north(p) = 0}, LRS4[2] = {p | p in GRS and south(p) = 0}, LRS4[3] = {p | p in GRS and east(p) = 0} and LRS4[4] = {p in GRS and west(p) = 0}.

Definition 9. **Skeleton**, SS, is a set of black points that can not be deleted by some thinning algorithms.

Definition 10. A skeleton S is of **one-pixel-wide** if all points in S are in IPS.

Definition 11. **Thinning** a binary image means application of an iterative procedure for removing locally removable points.

Definition 12. A thinning algorithm is **perfect** if it can generates one-pixel-wide skeletons.

3. Parallel Thinning with Subiterations

In parallel thinning, pixels are examined for deletion based on the results of only the previous iteration. For this reason, these algorithms are suitable for implementation on parallel processors where the pixels satisfying a set of conditions can be removed simultaneously. Unfortunately, fully parallel algorithms can have difficult preserving the connectedness of an image if only 3 * 3 window is considered. Therefore, the usual practice is to use 3 * 3 neighborhoods but to divide each iteration into subiteration in which only a subset of contour pixels are considered for removal. At the end of each subiteration, the remaining image is updated for the next subiteration [3]. For example, GH[1] is a 2-subiteration parallel thinning algorithm with one subiteration deleting the north and east contour points and the other deleting the south and west contour points. The subiteration conditions appear in the following.

```
a. CN(p) = 1
b. 2 <= N(p) <= 3
c. Apply one of the following:
   1. (p[0] v p[1] v ~p[3]) ^ p[2] = 0
        ----  (the first subiteration)
   2. (p[4] v p[5] v ~p[7]) ^ p[6] = 0
        ----  (the second subiteration)
```

Thinning stops when no further deletions occur. Where N(p) = MIN{N1(p), N2(p)}

```
N1(p) = (p[7] v p[0]) + (p[1] v p[2]) +
        (p[3] v p[4]) + (p[5] v p[6]) and
N2(p) = (p[0] v p[1]) + (p[2] v p[3]) +
        (p[4] v p[5]) + (p[6] v p[7])
```

3.1. WN-Notation of Thinning Conditions

In this section we propose a **WN-Notation Set** (Weight Number Notation Set) to represent the thinning condition C_k of n-subiteration algorithm A ($1 \le k \le n$):

$$A[k] = \{ WN(p) \mid WN(p) \in GRS \wedge p \in C_k \}$$

Therefore, the WN-Notation Sets of 2-subiteration algorithm GH can be constructed:

```
GH[1]={028,056,060,065,067,080,081,083,088,089,
       092,097,099,112,113,115,120,121,124,131,
       193,195,208,209,211,216,217,220,224,225,
       227,240,241,243,248,249,252} - 37

GH[2]={005,007,013,014,015,020,021,022,023,028,
       029,030,031,052,053,054,055,056,060,061,
       062,063,131,133,135,141,143,149,151,157,
       159,193,195,197,199,205,207} - 37
```

Using WN-Notation, the algorithm GH is like this.

```
procedure GH_A(Q, max_row, max_col);
begin
   thinning_not_end := TRUE;
   while (thinning_not_end) do
   begin
       thinning_not_end := FALSE;
       for k := 1 to 2 do
       begin
          WNQ:=WN1(Q);
          for row := 2 to max_row -1 do
          for col := 2 to max_col -1 do
          if (WNQ[row,col] in GH[k]) then
          begin
             thinning_not_end := TRUE;
             q[row,col] := 0;
          end;
       end;
   end;
end;
```

Where Q is the Pattern q[1..max_row, 1..max_col], "WNQ:=WN_1(Q)" means a weight number matrix WRQ[i, j] to be constructed by WN_1(q[i, j]) using window W_1, i from 1 to max_row, j from 1 to max_col. The set of all removable points for algorithm GH is GH_Remove (include two subiterations) as follows :

```
GH_Remove = GH[1] + GH[2] =
{ 005,007,013,014,015,020,021,022,023,028,029,030,
  031,052,053,054,055,056,060,061,062,063,065,067,
  080,081,083,088,089,092,097,099,112,113,115,120,
  121,124,131,133,135,141,143,149,151,157,159,193,
  195,197,199,205,207,208,209,211,216,217,220,224,
  225,227,240,241,243,248,249,252 } -- 68
```

The set of all redundant points for algorithm GH is:

```
   GH_Redundant = GRS - GH_Remove =
{ 003,006,012,024,048,069,071,077,079,084,086,091,
  094,096,101,103,109,111,116,118,123,126,129,181,
  183,189,191,192,212,214,219,222,229,231,237,239,
  244,246,251,254 } - 40
```

It means that only 68 types of pixels can be removed and the other 40 types of pixels can not be removed in GH thinning process. Therefore, Algorithm GH is not of one-pixel-wide.

3.2. Symmetry of Subiteration Conditions

In the section 3.1, two thinning condition sets, GH[1] and GH[2], are constructed for algorithm GH under the windows W_1 like this:

Subiteration	Window	Condition
1	W_1	GH[1]
2	W_1	GH[2]

If we construct the thinning condition set of the second subiteration of GH, i.e., GH[2], under the Window W_2 (see Figure 1), the thinning condition set GH[2] is the same as GH[1] like this:

Subiteration	Window	Condition
1	W_1	GH[1]
2	W_2	GH[1]

This is the symmetry of subiteration conditions. Algorithm GH[1], LW[4] and ZS[8] are all the examples with this property. Using this property, we can design new procedure for algorithm GH as follows, where weight number WN_k(p) is calculated by different window W_k (k = 1, 2).

```
procedure GH_B(Q, max_row, max_col);
begin
  thinning_not_end := TRUE;
  while (thinning_not_end) do
  begin
   thinning_not_end := FALSE;
   for k := 1 to 2 do
   begin
      WNQ:=WNk(Q)
      for row := 2 to max_row -1 do
      for col := 2 to max_col -1 do
      if (WNQ[row,col] in GH[1]) then
      begin
         thinning_not_end := TRUE;
         q[row,col] := 0;
      end;
   end;
  end;
end;
```

3.3. Thinning Conditions with Array Representation

As mentioned, there are only 256 weight numbers for 3*3 window. It is possible to use an array GhAlg[0..255] to represent the thinning condition set GH[1]: the value of GhAlg[t] is 1 if t is in GH[1], otherwise GhAlg[t] is 0 (t from 0 to 255). Using symmetry and array representation of thinning conditions, we can design algorithm GH as follows.

```
procedure GH_C(Q, max_row, max_col);
begin
   thinning_not_end := TRUE;
   while (thinning_not_end) do
   begin
       thinning_not_end := FALSE;
       for k := 1 to 2 do
       begin
          WNQ:=WNk(Q)
          for row := 2 to max_row -1 do
          for col := 2 to max_col -1 do
          if (GhAlg[WNQ[row,col]]) then
          begin
             thinning_not_end := TRUE;
             q[row,col] := 0;
          end;
       end;
   end;
end;
```

4. PTA2T: Proposed Algorithm

Figure 2 shows the templates for use in the proposed thinning algorithm. There are eleven 3*3 templates (T01 to T11) in this template set. The symbols 'p', '0', '1', '*' and '#' used in these templates denote the currently tested pixel, a white pixel, a black pixel, a don't-care pixel, and a pixel in IPS respectively.

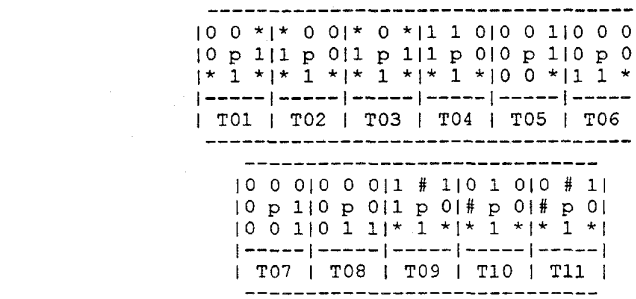

Figure 2. Proposed thinning templates

We can convert all templates to WN-notation as follows

```
T01 = { 20, 22, 28, 30, 52, 54, 60, 62}  -- 8
T02 = { 80, 88,112,120,208,216,240,248}  -- 8
T03 = { 84, 86, 92, 94,116,118,124,126,212,214,
        220,222,244,246,252,254}  -- 16
T04 = {209,217,241,249}  -- 4
T05 = {  6, 14}  -- 2
T06 = { 48, 56}  -- 2
T07 = { 12}  -- 1
T08 = { 24}  -- 1
T09 = {211,219,243,251}  -- 4
T10 = { 81, 89,113,121}  -- 4
T11 = { 83, 91,115,123}  -- 4
```

```
Pta2t[1] = T1 + T2 + T3 + T4 + T5 + T6
                + T7 + T8 + T9 + T10 + T11 =
      {006,012,014,020,022,024,028,030,048,052,
       054,056,060,062,080,081,083,084,086,088,
       089,091,092,094,112,113,115,116,118,120,
       121,123,124,126,208,209,211,212,214,216,
       217,219,220,222,240,241,243,244,246,248,
       249,251,252,254} -- 54
```

In order to distinguish the templates, we use the following values to fill the array: Pta2tAlg[0..255]:

Array Representation		
Pta2tAlg [WN(p)]	p is stable	
0	no	
1	yes	(T01 - T08)
2	yes if north(p) is in IPS	(T09)
3	yes if west(p) is in IPS	(T10)
4	yes if north(p) and west(p) are in IPS	(T11)

The algorithm array Pta2tAlg[0..255] can be constructed as follows.

```
Pta2tAlg[0..255] =
      {0,0,0,0,0,0,1,0,0,0, 0,0,1,0,1,0,0,0,0,0,
       1,0,1,0,1,0,0,0,1,0, 1,0,0,0,0,0,0,0,0,0,
       0,0,0,0,0,0,0,0,1,0, 0,0,1,0,1,0,1,0,0,0,
       1,0,1,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,
       1,3,0,4,1,0,1,0,1,3, 0,4,1,0,1,0,0,0,0,0,
       0,0,0,0,0,0,0,0,0,0, 0,0,1,3,0,4,1,0,1,0,
       1,3,0,4,1,0,1,0,0,0, 0,0,0,0,0,0,0,0,0,0,
       0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,
       0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,
       0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,
       0,0,0,0,0,0,0,0,1,1, 0,2,1,0,1,0,1,1,0,2,
       1,0,1,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,
       1,1,0,2,1,0,1,0,1,1, 0,2,1,0,1,0}
```

Theorem 1. Algorithm PTA2T is perfect.

Proof. We already constructed Pta2t[1] above. In the same way, we can construct Pta2t[2] as follows.

```
Pta2t[2] =
    { 003,005,007,013,015,021,023,029,031,053,
      055,061,063,065,067,069,071,077,079,096,
      097,099,101,103,109,111,129,131,133,135,
      141,143,149,151,157,159,181,183,189,191,
      192,193,195,197,199,205,207,224,225,227,
      229,231,237,239 } - 54
```

The total deleted types of pixels in algorithm PTA2T thinning process is Pta2t[1] + Pta2t[2] which is equal to GRS. Therefore, algorithm PTA2T is perfect because all pixels in GRS are deleted.

5. Comparisons of Algorithms

To compare the different thinning algorithms with 2 subiterations, we design a procedure called **ThinAlg** shown in the following. There are four parameters for this procedure: Q is digitized pattern array; max_row is the maximum row number of pattern; max_col is the maximum column number of pattern; AlgArr is the algorithm array which can be Pta2tAlg or GhAlg or ZsAlg. The image and results of the tested example are shown in Figure 3 and Figure 4. These experimental results show that the proposed algorithm is of one-pixel-wide and faster than algorithm GH and ZS.

```
procedure ThinAlg(AlgArr, Q, max_row, max_col);
begin
  thinning_not_end:= TRUE;
  while (thinning_not_end) do
  begin
    thinning_not_end:= FALSE;
    for k:=1 to 2 do
    begin
      WNQ := WNk(Q);
      for row:=2 to max_row-1 do
      for col:=2 to max_col-1 do
      begin
        stable:=FALSE;
        win := Wk(WNQ[row,col]);
        case AlgArr[WNQ[row,col]] of
        1: stable:= TRUE;
        2: if AlgArr[win[0]]=0 then stable:= TRUE;
        3: if AlgArr[win[6]]=0 then stable:= TRUE;
        4: if AlgArr[win[0]]=0 and AlgArr[win[6]]=0
             then stable:=TRUE;
        end; {case}
        if (stable) then
        begin
          q[row,col]:=0;
          thinning_not_end:=TRUE;
```

```
        end;
      end; {row, col}
    end; {k}
  end; {while}
end; {ThinAlg}
```

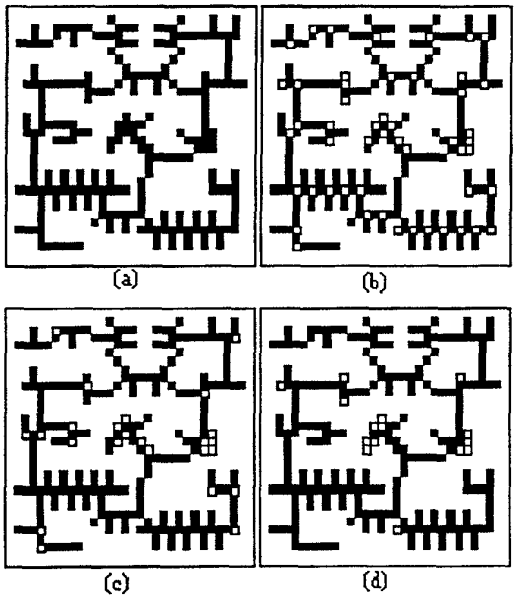

Figure 3. Comparisons of thinning results
(the white rectangle is the removed pixel)
(a). Original image. (b). Skeleton by PTA2T.
(c). Skeleton by GH. (d). Skeleton by ZS

Iter-ation	Sub-iter-ation	Tested Pixel PTA2T	Tested Pixel GH	Tested Pixel ZS	Removed Pixel PTA2T	Removed Pixel GH	Removed Pixel ZS	
1	1	261	261	261	38	16	12	(A)
1	2	223	245	249	13	7	2	
2	1	210	238	247	0	1	1	
2	2	210	237	246	0	0	0	
3	1		237	246		0	0	
3	2		237	246		0	0	

Algorithm	Total Tested Pixel	Total Removed Pixel	Redundant Pixel	
PTA2T	904	51	0	(B)
GH	1455	24	27	
ZS	1499	15	36	

Figure 4. Comparisons of 3 thinning algorithms

6. Discussions

We propose a fast 2-subiteration parallel thinning algorithm (PTA2T) which preserves the connectivity of patterns and produces one-pixel-wide skeletons. Using the same idea, we can design a fully parallel thinning algorithm that generates skeletons without redundant pixels.

Reference

1. Gao, Z. and Hall, W. R., "Parallel Thinning with Two- Subiteration Algorithm", *CACM*, March 1989, Vol.32, No.3, pp.359-373.

2. Jang, B. K. and Chin, R. T., "Analysis of thinning algorithms using mathematical morphology", *IEEE Trans. Pattern Anal. Mach. Intell.* PAMI-12, 6 (1990) 541-551.

3. Lam, L., Lee, S. W. and Suen, C. Y., "Thinning methodologies - a comprehensive survey", *IEEE Trans. Pattern Anal. Mach. Intell.* 14, 9 (1992) 869-885.

4. Lu, H. E. and Wang, P. S. P., 'A Comment on "A Fast Parallel Algorithm for Thinning Digital Patterns"', *CACM*, 29, No.3, March 1986, pp.239-242.

5. Rosenfeld, A., "A characterization of parallel thinning algorithms", *Inform. Contr.*, Nov. 1975, Vol. 29, pp. 286-291.

6. Rosenfeld, A. and Davis, L. S., "A note on thinning", *IEEE Trans. on System, Man, and Cybern.*, March 1976, Vol. SMC-6, No.3, pp.226-228.

7. Tamura, H., "A comparison of line thinning algorithms from digital geometry viewpoint", *Proc. 4th Int. Conf. Pattern Recognition*, Kyoto, pp. 715-719 (1978).

8. Zhang, T. Y. and Suen, C. Y. "A Fast Parallel Algorithm for Thinning Digital Patterns", *CACM* 27 (1984) 236-239.