Image Processing and Computer Vision

Robert Haralick Editor

An Improved Parallel Thinning Algorithm

CHRISTOPHER M. HOLT, ALAN STEWART, MAURICE CLINT, and RONALD H. PERROTT

ABSTRACT: An iterative thinning algorithm reduces a two-dimensional pattern of strokes to its skeleton by removing layers of edge elements until each stroke has unit thickness. A parallel solution requires the independent calculation of new values for each iteration, using a window of nearest neighbors for each element. The traditional need for at least two subiterations can be avoided by modifying the window to permit the availability of intermediate calculations. Timings on an ICL DAP (an array processor) indicate an improvement of over 40 percent. Additional refinements are suggested to reduce noise in the final skeleton.

1. BACKGROUND

A digital picture of strokes (bands of elements) can be represented as a logical matrix of true and false elements (1s and 0s). A thinning algorithm reduces each stroke to a skeleton of unit thickness, facilitating pattern recognition. The reduction should not introduce unnecessary branches or deviate significantly from the centers of strokes: the outer layers of a stroke are to be peeled off as clearly as possible until no elements can be removed by further stripping.

Most algorithms [1–3] assume a 3 × 3 window for each picture element. That is, the values of the eight neighbors of a central element (C) are used in the calculation of its value for the next iteration. The neighboring values can be denoted by the compass directions (north, south, east, etc.).

NW	N	NE
W	C	E
SW	S	SE

An element is to be removed when it is on the edge of a stroke. This is often identified with the condition of having from two to six connected neighbors, where connectedness is defined as the existence of a path between any two true neighbors that passes only through true neighbors. The function "edge" (Appendix A) returns the value true when this condition is satisfied. An element is preserved if it is present at the beginning of an iteration and if the conditions for removal are not met. Thus, a first approximation for determining the status of an element is the logical expression

$$v\text{C And} \sim \text{edgeC}$$

(where ~ denotes Not, and v denotes the value at a location). Additional conditions are necessary to ensure that a parallel algorithm satisfies the following constraints:

a) The removal of the edges of a stroke should not remove the entire stroke (e.g., a vertical stroke of width 2 should not disappear).
b) All four elements of an isolated 2 × 2 square should not be removed.
c) The connectedness of the overall structure should not be affected by the simultaneous removal of a number of elements.

This research was supported by the Science and Engineering Council under grant GR/B/42343.

A common solution is to introduce a directional bias, for example, by favoring north over south, and west over east. Distortion is minimized by introducing subiterations that differ only in the directions of the bias. For example, with two subiterations, the first may permit the removal of an element if it is on a south or east edge or if it is on a corner [3]. This is described by the logical expression

$$\sim v\text{E Or } \sim v\text{S Or } (\sim v\text{N And } \sim v\text{W}).$$

The expression for survival then includes the negation of this, combined with the preservation expression above:

$$v\text{C And } (\sim\text{edgeC Or } (v\text{E And } v\text{S And } (v\text{N Or } v\text{W})).$$

The second subiteration interchanges north with south and east with west, associating with the element C the value (see Figure 1)

$$v\text{C And } (\sim\text{edgeC Or } (v\text{W And } v\text{N And } (v\text{S Or } v\text{E})).$$

				1a				
			1a	1b	1a			
		1a	X	X	X	1a		
	1a	X	1a		X	1b	1a	
	1b	X	1a		1b	X	1a	
1a	1b	X	X	X	X	X	1b	1a
1b	X	1b	1a	1a	1a	1b	X	1a
1b	X	1a				1b	X	1a
1b	X	1a				1b	X	1a
1b	1b	1a				1b	1b	1a
1a	1a	1a				1a	1a	1a

FIGURE 1. The Effect Of The Zhang and Suen Algorithm [3]. 1a denotes a pixel removed in the first subiteration. 1b denotes a pixel removed in the second subiteration. X denotes a pixel in the final skeleton.

2. MODIFYING THE ALGORITHM

The survival of a particular element need not be determined using only the values directly accessible in the 3 × 3 window. Only one subiteration is necessary if edge information about neighbors is made available. The additional cost in communication time is very small.

A vertical stroke of width 2 (a 2-stroke) is guarded by keeping one of its edges. Maintaining the westward bias of the original algorithm, an element on a west edge is preserved if it is not on a corner and its east neighbor is on an edge, that is, if

$$\text{edgeE And } v\text{N And } v\text{S}$$

is true. Similarly, the north edge of a horizontal 2-stroke is preserved if the expression

$$\text{edgeS And } v\text{E And } v\text{W}$$

is true. These conditions ensure that connectedness is preserved (Appendix B).

The removal of each element of a 2 × 2 square can be prevented by checking the east, south, and southeast neighbors. If each of these is on a corner the expression

$$\text{edgeE And edgeSE And edgeS}$$

is true, and the element is to be preserved.

An element survives an iteration if the combined conditions represented by the expression

$$\begin{aligned} v\text{C And } (\sim\text{edgeC Or}\\ &(\text{edgeE And } v\text{N And } v\text{S) Or}\\ &(\text{edgeS And } v\text{W And } v\text{E) Or}\\ &(\text{edgeE And edgeSE And edgeS})) \end{aligned}$$

is true. The use of this expression leads to a skeleton almost identical to that resulting from the original, 2-pass algorithm [3]. See Figure 2.

				1				
			1	2	1			
		1	X	X	X	1		
	1	X	1		1	X	1	
	1	X	1		1	X	1	
1	2	X	X	X	X	X	2	1
1	X	2	1	1	1	2	X	1
1	X	1				1	X	1
1	X	1				1	X	1
1	X	1				1	X	1
1	1	1				1	1	1

FIGURE 2. The Effect Of The Modified Algorithm. 1 denotes a pixel removed in the first subiteration. 2 denotes a pixel removed in the second subiteration. X denotes a pixel in the final skeleton.

3. REDUCING NOISE

For many cases the above algorithm is adequate for thinning a pattern. However, there are occasions when it is worth taking extra time to reduce distortion. One approach is to modify the edge conditions leading to the removal of elements. For example, an element may sometimes be removed when it has seven neighbors, rather than from two to six.

If an element is missing a single side neighbor (N, S, E, W), then the overall connectedness of the picture is unaffected by the removal of the element. This case can be incorporated into the edge function by modifying the tests on neighbors, resulting in a slightly faster algorithm.

If, within an iteration, an element is on an edge after ordinary edge elements have been removed, then it may be removed without affecting overall connectedness. The restriction is imposed that such elimination is permitted only for those elements that had at least one empty neighbor at the beginning of the iteration. The elements removed include those that were missing a single corner neighbor (NW, NE, SW, SE). The effect, unlike that resulting from another iteration, is to reduce distortions in concave boundaries and smooth the removal of diagonal edges.

To resolve conflicts where second-level edge elements are adjacent, a directional bias must be imposed just as for primary edge elements. It is simplest to copy the primary conditions, introducing only the check for the initial presence of empty neighbors. The value of the expression

$$\begin{aligned} &v'\text{C And } (\sim\text{edge}'\text{C Or } \sim t01(\text{C}) \text{ Or}\\ &\quad(\text{edge}'\text{S And } v'\text{W And } v'\text{E}) \text{ Or}\\ &\quad(\text{edge}'\text{E And } v'\text{N And } v'\text{S}) \text{ Or}\\ &\quad(\text{edge}'\text{S And edge}'\text{E And edge}'\text{SE})) \end{aligned}$$

then determines the survival of an element (a prime denotes the second half of an iteration and $\sim t01$ is the check for at least one absent neighbor (Appendix A)). See Figure 3.

```
                1a
            1a  X   1a
        1a  X   1a  X   1a
    1a  X   1a      1a  X   1a
    1a  X   1a      1a  X   1a
1a  1b  X   X   X   X   X   1b  1a
1a  X   1b  1a  1a  1a  1b  X   1a
1a  X   1b              1a  X   1a
1a  X   1a              1a  X   1a
1a  X   1a              1a  X   1a
1a  1a  1a              1a  1a  1a
```

FIGURE 3. The Effect Of The Extended Modified Algorithm. 1a denotes a pixel removed in the first subiteration. 1b denotes a pixel removed in the second subiteration. X denotes a pixel in the final skeleton.

4. STAIRCASE ELIMINATION

When edges have been removed as far as is possible, unnecessary skeletal elements may remain which cannot be removed using the algorithms above. For example, with the pattern

$$\begin{matrix} 0 & 1 & 0 & 0\\ 0 & 1 & 1 & 0\\ 0 & 0 & 1 & 0 \end{matrix}$$

either element of the middle line may be removed without changing the connectedness of the overall picture. To cope with such a case, a final "clean-up" phase can be introduced after edge removal.

Overall connectedness is guaranteed by ensuring that an element is removed only if it has two adjacent side neighbors with the nonadjacent corner absent. Maintaining an initial northward directional bias, this means that an element can be removed if it has a window of the form

$$\begin{matrix} 0 & 1 & x\\ 1 & 1 & x\\ x & x & 0 \end{matrix} \quad\text{or}\quad \begin{matrix} x & 1 & 0\\ x & 1 & 1\\ 0 & x & x \end{matrix}$$

where the values of the unspecified neighbors may be either 0 or 1. The side neighbors that must be present cannot be removed because their windows match neither of the two patterns.

To ensure that a hole is not created, the condition is added that one of the unknown side neighbors must be absent. Combined with the conditions above, the overall expression indicating survival is

$$\begin{aligned} &v\text{C And } \sim(v\text{N And}\\ &\quad((v\text{E AND } \sim v\text{NE And } \sim v\text{SW And}\\ &\qquad(\sim v\text{W Or } \sim v\text{S})) \text{ Or}\\ &\quad(v\text{W And } \sim v\text{NW And } \sim v\text{SE And}\\ &\qquad(\sim v\text{E Or } \sim v\text{S}))). \end{aligned}$$

After elements have been removed using the northward bias above, it is necessary to repeat the operation using a southward bias. Exactly the same expression is used except that north and south are interchanged.

5. COMPLEXITY OF THE ALGORITHM

It is assumed that one processor is provided for each element. The time for shifting a value from one position to that of a neighbor is s and that for performing an assignment or a logical operation is o. Execution of an If-Then statement is assumed to require one logical operation.

The edge function requires $85o + 8s$ units of time. The algorithm based on two subiterations [3] requires $7o$ additional units to determine the new status of an element. One complete iteration requires twice the sum of these times, for a total of $184o + 16s$ units of time per iteration.

The simplest form of the new algorithm requires an additional cost of $12o + 3s$ after the edge condition has been computed. Thus, the total time is $97o + 11s$ units per iteration.

The time required for the removal of elements with seven neighbors has two components. Permitting the removal of elements with a single missing side reduces the time by $24o$ units. However, the "reintroduction" of a second subiteration means that edge must be calculated twice, with an overall cost of $148o + 22s$ units. This is comparable to the original time, though the number of iterations may be reduced.

Staircase elimination requires $21o + 8s$ units for each of the north and south subiterations. As very few such iterations are ever needed the cost of introducing this improvement is low.

6. IMPLEMENTATION

The algorithm has been implemented on an ICL DAP, a 64 × 64 array processor. A typical result is shown in Figure 4. The original Zhang and Suen algorithm was modified as described above, first without and then including the tests for removing elements with seven neighbors. Four test patterns of varying complexity were used; the results for one, given below, are typical. They include the total time used, the number of iterations for a simple pattern, and the time per iteration (a constant over any kind of pattern). Times are given in milliseconds; the total time includes an initial overhead of 1.58 milliseconds.

Algorithm	Time	Iterations	Time / Iterations
Zhang and Suen	4.74	8	.39
New without 7 neighbor tests	2.84	6	.21
New with 7 neighbor tests	4.14	6	.43

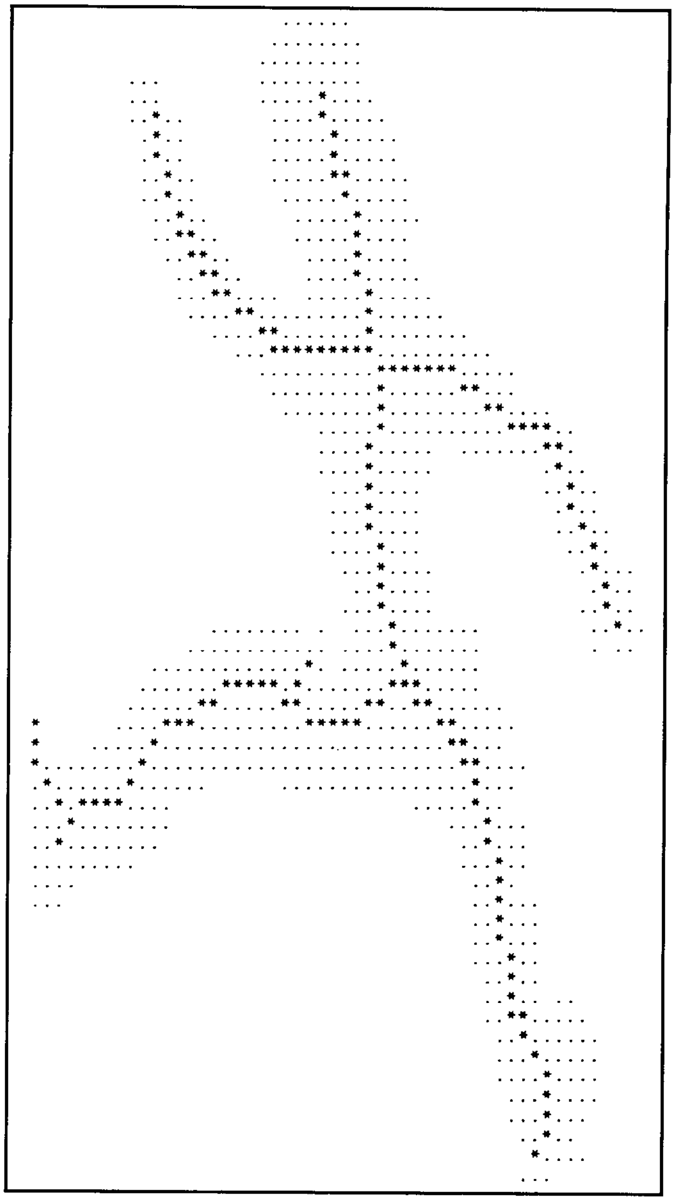

FIGURE 4. The Result Of Applying The Modified Algorithm Discussed In The Summary

The basic improvement in time is 40 percent, partly due to the reduced time per iteration and partly due to the reduced number of iterations. Even when seven neighbor tests are included and the time per iteration is higher than the original, the overall time is reduced because fewer iterations are needed.

7. SUMMARY

It is possible to remove the need for alternating subiterations of a thinning algorithm by expanding the window for each element to include edge information about its neighbors. This reduces the time required for thinning substantially. This assertion has been confirmed by tests on an ICL DAP, a distributed array processor in which neighboring processors are able to communicate in two dimensions. Further refinements are also proposed in the algorithm to yield cleaner skeletons, though a time penalty must be paid.

APPENDIX A
THE EDGE FUNCTION

The basic approach in implementing the edge function is to take pairs of neighbors around the center. For example, (vNW, vN) is a pair; the next pair is (vN, vNE); and so on. There are between two and six connected neighbors when at least one (0, 0) pair is found, at least one (1, 1) pair is found, and exactly one (0, 1) pair is found. To minimize logical operations, these tests are performed on one side at a time. It is assumed that 0 denotes False and 1 denotes True. The local variables in the text below take the value True if their specified pairs have been found; for example, t00 is True if a (0, 0) pair has been found. t01s is assigned the value True if more than one (0, 1) pair is present.

```
Function edge: Boolean;
  Var t00, t01, t01s, t11: Boolean;

  Procedure check(v1, v2, v3:Boolean);
    Begin
      If ~v2 And (~v1 Or ~v3) Then t00 := True;
      If v2 And (v1 Or v3) Then t11 := True;
      If (~v1 And v2) Or (~v2 And v3) Then
        Begin t01s := t01; t01 := True End
    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 ~t01s;
  End;
```

This pseudocode can be translated easily into DAP or standard Fortran. The function can be adapted to permit the inclusion of elements with seven neighbors and a missing side by eliminating t00 and modifying the final line of edge to be

```
edge := vC And t11 And t01 And ~t01s
                And ~(vN And vE And vS And vW).
```

APPENDIX B
PRESERVING CONNECTEDNESS

It is necessary to ensure that the connectedness properties of the overall picture are preserved. Given between two and six connected neighbors, there must be at least one side neighbor present and at least one missing.

If only one side neighbor is present, at least one corner must be present. Then, the side neighbor has unconnected neighbors, and cannot be removed.

$$\begin{matrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ x & 1 & x \end{matrix}$$

The center element may therefore be removed.

If two side neighbors are present, they must be neighbors of one another (by connectedness).

$$\begin{matrix} x & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & x \end{matrix}$$

If an x is 0, then removing the center and the adjacent side does not affect the overall connectedness. If an x is 1, then the adjacent side cannot be removed: either only one of its neighbors is absent or its neighbors are not connected.

If there are three sides present, the above argument applies to those sides adjacent to the empty side (east and west in the diagram).

$$\begin{matrix} x & 0 & x \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$$

The only way to disrupt connectedness is to remove the remaining side (south in the diagram); but the conditions preventing the removal of 2-strokes are sufficient to prevent both the center and that side from being removed at once. This last case ensures that connectedness is maintained when elements are to be removed that have seven neighbors and a missing side.

REFERENCES

1. Hilditch, C.J. Linear skeletons from square cupboards. In *Machine Intelligence IV*, B. Meltzer and D. Michie, Eds., American Elsevier, New York, 1969, 403–420.
2. Stefanelli, R., and Rosenfeld, A. Some parallel thinning algorithms for digital pictures. *J. ACM 18*, 2 (Apr. 1971), 255–264.
3. Zhang, T.Y., and Suen, C.Y. A fast parallel algorithm for thinning digital patterns. *Commun. ACM 27*, 3 (Mar. 1984), 236–239.

CR Categories and Subject Descriptors: D.1.3 [**Programming Techniques**]: Concurrent Programming; I.5.2 [**Pattern Recognition**]: Design Methodology—*pattern analysis*; I.5.4 [**Pattern Recognition**]: Applications—*computer vision*
General Terms: Algorithms, Design, Theory
Additional Key Words and Phrases: parallel algorithms, skeletonization, thinning of digital patterns

Received 3/85; revised 11/85; accepted 8/86

Authors' Present Addresses: Christopher M. Holt, Alan Stewart, Maurice Clint, and Ronald H. Perrott, Department of Computer Science, Queen's University of Belfast, Belfast, Northern Ireland BT7 1NN.