

Parallel Thinning with Two-Subiteration Algorithms

ZICHENG GUO and RICHARD W. HALL

ABSTRACT: Two parallel thinning algorithms are presented and evaluated in this article. The two algorithms use two-subiteration approaches: (1) alternatively deleting north and east and then south and west boundary pixels and (2) alternately applying a thinning operator to one of two subfields. Image connectivities are proven to be preserved and the algorithms' speed and medial curve thinness are compared to other two-subiteration approaches and a fully parallel approach. Both approaches produce very thin medial curves and the second achieves the fastest overall parallel thinning.

1. INTRODUCTION

Thinning is a fundamental preprocessing step in many image processing and pattern recognition algorithms. When the fundamental primitives in an image are strokes or curves of varying thickness it is usually desirable to reduce them to thin representations located along the approximate middle of the original stroke or curve. Such thinned representations are typically easier to process in later stages producing savings in both time and storage complexity. In some approaches the thinned result is intended to contain sufficient information to achieve recovery of the original image [1, 3, 11, 12], but investigators have frequently focused solely on the reductive aspect of thinning since image recovery is not always necessary. Early thinning approaches [8, 18] were designed for serial implementation, but there has been a growing interest in parallel thinning algorithms over the past two decades as parallel image processing structures have become more available [13]. In these structures local operators are most efficiently computed if their supports are smaller and interest has focused on supports defined over the 3×3 neighborhood of a pixel. Fully parallel thinning algorithms which are restricted to operators with 3×3 support have difficulty preserving the connectivity of an image. (This is discussed in Section 2) Thus, investigators have partially serialized their algorithms by breaking a given iteration of their algorithms into several distinct iterations (referred to as subiterations) [2, 10, 15–17, 19–21] or by

partitioning the image space into distinct subfields [6, 13]. Either technique has the effect of defining distinct subiterations which tend to apply thinning operators to different parts of an image. Thus we refer to both approaches as thinning algorithms with subiterations. When using either technique with 3×3 local operators the optimal choice for minimizing number of subiterations will usually be two subiterations. The focus of this article is on two-subiteration approaches which are restricted to operators with 3×3 support. More recently, however, fully parallel thinners with effective supports larger than 3×3 have been presented [4, 9] and will be addressed in Section 5.

Particularly fast two-subiteration approaches using 3×3 supports have been reported [10, 21]. Although the parallel speed (e.g., required number of iterations) of these algorithms was not reported, we have confirmed that both do outperform the original two-subiteration approach [19]. Unfortunately, these approaches do not always preserve connectivity in images and do not always produce thin results. Our goal is to find fast two-subiteration algorithms restricted to 3×3 thinning operators which preserve connectivity in images and produce thin results. In this article we present two new approaches to parallel thinning: a two-subiteration approach modified from [10, 21] to preserve connectivity in all images and produce thinner results and a two-subiteration approach using subfields which achieves for almost all of our test cases the fastest parallel thinning reported to date. We compare our algorithms to other two-subiteration algorithms reported [10, 19, 21] by considering preservation of connectivity, parallel speed, and thinness and size of results. In our evaluations we use artificial and natural images including thick lines of various orientations, Chinese characters and English letters. Finally we consider two recently reported fully parallel thinners that use a support larger than 3×3 [4, 9].

2. PARALLEL THINNING IN RECTANGULAR TESSELLATIONS

We will define our thinning algorithms over binary valued (0 or 1) images digitized in a rectangular tessellation. We imagine that 1-valued pixels (ones) form a set,

S , representing objects (connected components) to be thinned; and 0-valued pixels (zeros) form a set, \bar{S} , representing either the background of or holes in S . To avoid connectivity paradoxes we define connectivities for S and \bar{S} with eight-connectivity and four-connectivity, respectively. After thinning is complete the thinned objects should be a curve or a union of curves which we will refer to as the medial curves. A set of pixels, G , is curve-like if most of the pixels of G have exactly two eight-neighbors in G and a few pixels in G are end-points (with one eight-neighbor in G) or branch points (more than two eight-neighbors in G) [16].

Specific goals for any thinning algorithm have been defined by Rosenfeld [15, 16]. We paraphrase and extend these goals for parallel thinners:

1. Connectivity is preserved for both S and \bar{S} .
2. Objects which are already curves or unions of curves should be unchanged.
3. Medial curves should
 - a. Lie along the approximate midpoints of elongated objects
 - b. Be as thin as possible.
4. A parallel thinner should require as few iterations as possible.

Goal 1 can be satisfied absolutely while goals 2 and 3 are satisfied to only varying degrees by various algorithms. Goal 4 states the desire for a high parallel speed as measured by the number of required iterations. For our new algorithms we will show that goal 1 is satisfied, discuss qualitatively how well our medial curve results satisfy goals 2 and 3a, and provide quantitative measures to compare the thinness of results (goal 3b) and the parallel speed (goal 4).

We will focus primarily on algorithms with thinning operators which use 3×3 supports which include a pixel and its eight nearest neighbors. This is a natural restriction since many parallel computer structures imagine that an image is placed in a rectangular array of processing elements (PEs) with one pixel per PE and with each PE linked to its nearest neighbors (e.g., see [13]). But it is known that certain problems arise for fully parallel thinning or shrinking algorithms which use reduction only 3×3 operators [14–17]. For example, if we define a parallel thinning algorithm with identical thinning results over 90° rotations of the object we will typically completely delete the 2×2 square object; thus violating goal 1 [15]. Furthermore, we know that a long, three-width horizontal rectangle with length $O(n)$ can be thinned in $O(1)$ iterations using, say, the four-subiteration thinner of Rosenfeld [15]. But if we are thinning fully in parallel, in order to avoid requiring $O(n)$ iterations we must delete the inner north and south boundary ones of this long rectangle. This same operator applied to a long, horizontal rectangle of width two will always disconnect or completely delete this object. Thus, fully parallel thinning algorithms that use reduction only 3×3 operators are unable to meet our thinning goals.

To avoid these problems when using 3×3 supports investigators have partially serialized their otherwise parallel algorithms. Two classes of approaches have been used where one fully parallel iteration operating over all pixels is:

1. Broken into i distinct iterations (subiterations) each of which tends to apply a thinning operator to a subset of the pixels in the image, usually using distinct boundary conditions at each subiteration [2, 10, 15–17, 19–21] or
2. Broken into i distinct iterations of the same parallel operator applied over i distinct subfields of the image [6, 13].

Subiterations with $i = 2$ [10, 16, 19–21], $i = 4$ [15–17, 19, 20] and $i = 8$ [3] have been applied in rectangular tessellations and subfields with $i = 3$ and 4 have been utilized for hexagonal [6, 13] and rectangular tessellations [13], respectively. The parallel speed of a parallel algorithm is measured by the number of required iterations which is simply the number of executed iterations for a fully parallel algorithm, the number of executed subiterations for a “subiterations” algorithm, or the number of parallel applications of an operator over a subfield for a “subfields” algorithm. Thus, the number of subiterations or subfields used directly affects the number of total iterations required to thin the image. In a parallel implementation (particularly where all competing thinning operators are implementable in a similar amount of time) the number of required iterations is a key measure of time complexity for the thinning algorithms. If sufficient hardware is available to allow the computation of any logical function over a given support, then parallel speed measures identify the ultimately fastest algorithms achievable over the given support. When thinning operators with 3×3 support are used, making fully parallel algorithms out of the question, then the optimally fast choice is for two-subiteration approaches.

3. A TWO-SUBITERATION THINNING ALGORITHM

We have modified the two-subiteration algorithms presented in [21] and improved in [10] to preserve connectivity properties and to produce thinner results. Our algorithm uses operators with a 3×3 support as defined in Figure 1. We refer to p_2, p_4, p_6 , and p_8 as p 's side neighbors and p_1, p_3, p_5 , and p_7 as p 's diagonal neighbors. We define several variables over this support to be used in this algorithm. $C(p)$ is defined as the number of distinct eight-connected components of ones in p 's eight-neighborhood. $C(p) = 1$ implies p is eight-simple when p is a boundary pixel [17]. We define $B(p)$ as the number of ones in p 's eight-neighborhood. We use symbols $\bar{}$, \wedge and \vee to refer to logical complement, AND and OR, respectively; and reserve $+$ and \cdot for arithmetic addition and multiplication. We introduce a new variable, $N(p)$, which is useful for endpoint detection, but which can also help to achieve thinner results:

$$N(p) = \text{MIN}[N_1(p), N_2(p)] \quad (1)$$

where

$$N_1(p) = (p_1 \vee p_2) + (p_3 \vee p_4) + (p_5 \vee p_6) + (p_7 \vee p_8) \quad (2)$$

and

$$N_2(p) = (p_2 \vee p_3) + (p_4 \vee p_5) + (p_6 \vee p_7) + (p_8 \vee p_1). \quad (3)$$

$N_1(p)$ and $N_2(p)$ each break the ordered set of p 's neighboring pixels into four pairs of adjoining pixels and count the number of pairs which contain 1 or 2 ones.

Algorithm 1 (A1)

A one of S , p , is deleted (one changed to zero) iff all of the following conditions are satisfied:

- a. $C(p) = 1$;
- b. $2 \leq N(p) \leq 3$; and
- c. Apply one of the following:
 1. $(p_2 \vee p_3 \vee \bar{p}_5) \vee p_4 = 0$ in odd iterations, or
 2. $(p_6 \vee p_7 \vee \bar{p}_1) \wedge p_8 = 0$ in even iterations.

Thinning stops when no further deletions occur.

Condition a is a necessary condition for preserving local connectivity when p is deleted and avoids deletion of pixels in the middle of medial curves. Our use of $C(p)$ allows some of the ones in the middle of two-width diagonal lines to be deleted which in [10, 21] were preserved. Figure 2(a) shows one such case. In [10, 21] deletion of a pixel, p , requires that there be exactly one four-connected component of ones in p 's eight-neighborhood. Using this rule in Figure 2(a), p and b are each preserved; but p or b could be deleted to obtain a thinner result. In odd iterations Algorithm 1 allows the deletion of p since $C(p) = 1$. Similarly, in even iterations Algorithm 1 allows the deletion of b .

The new variable $N(p)$ provides an endpoint check replacing $B(p)$ which is used in [10, 21]. When $B(p) = 1$, p is an obvious endpoint and $N(p) = 1$. But when $B(p) = 2$, p may or may not be an endpoint. In Fig-

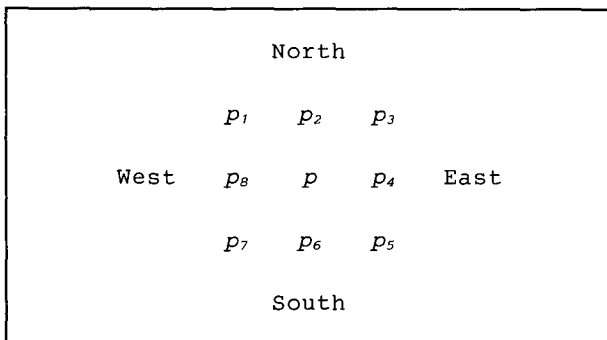


FIGURE 1. Neighborhood Definitions for Pixel p

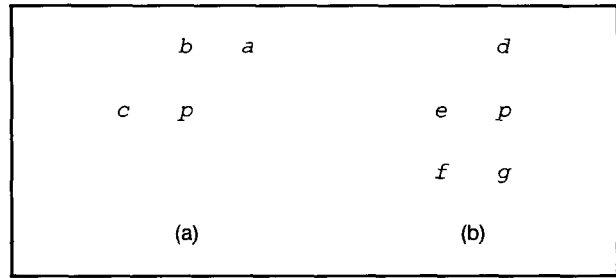


FIGURE 2. Objects Illustrating the Rationale for Conditions Used in Algorithm 1 (See Text)

ure 2(a) a count of neighboring ones will not discriminate between the redundant midpoint pixel p and the endpoints a or c . But, $N(a) = N(c) = 1$ while $N(p) = 2$. Thus, the redundant pixel p is deleted in odd iterations of A1 while endpoints a and c are preserved. In general our definition of $N(p)$ allows endpoints to be preserved while deleting many redundant pixels in the middle of curves.

Condition c(1), used for odd iterations, is satisfied when p 's neighborhood takes either of the forms shown in Figure 3, where x refers to a "don't care" pixel (either zero or one is acceptable). Condition c(2) is satisfied for 180° rotations of either of the two conditions shown in Figure 3. Thus, c(1) tends to identify pixels at the north and east boundary of objects and c(2) identifies pixels at the south and west boundary of objects. The condition in Figure 3b is more stringent than a simple north boundary check in order to preserve connectivity for objects like those shown in Figure 2. In an odd iteration pixels p and g are deleted while pixels a and d are preserved. Although pixels b and e are north border pixels, they must be preserved to maintain connectivity. The condition of Figure 3b enables preservation of both of these pixels.

Thinning of an object proceeds by successively removing in distinct iterations north and east, and then south and west boundary pixels. The connectivity properties of the image are preserved. Objects of S are neither disconnected nor are originally disjoint objects connected and no object can be completely deleted. The connectivity of \bar{S} is similarly preserved. Proofs for these assertions are given in Appendix A.

$C(p)$ in condition a can be evaluated efficiently by computing:

$$C_s(p) = \bar{p}_2 \wedge (p_3 \vee p_4) + \bar{p}_4 \wedge (p_5 \vee p_6) + \bar{p}_6 \wedge (p_7 \vee p_8) + \bar{p}_8 \wedge (p_1 \vee p_2). \quad (4)$$

$C_s(p)$ takes on values over the range [0, 4] as it counts the number of occurrences of a side zero with a one in at least one of the two adjacent pixels in the clockwise direction around p 's eight-neighborhood. It is shown in Appendix A that $C_s(p) = C(p)$ for all neighborhoods of p except cases where p has four side ones. In these cases

x	x	x		x	0	0
x	p	0		x	p	x
x	x	x		x	x	1
(a)				(b)		

FIGURE 3. Neighborhood Conditions Required for Deletion of p in Odd Iterations of Algorithm 1 (Symbol x Represents a Pixel which May Be Either 0 or 1)

$N(p) = 4$ making p not deletable by A1 for these cases. Thus, $C_s(p)$ may be used to compute condition **a** in A1.

4. A TWO-SUBITERATION THINNING ALGORITHM USING SUBFIELDS

A partial serialization of a parallel thinning algorithm can also be achieved by dividing the image into distinct subsets referred to as subfields. These subfields are activated sequentially in distinct iterations. This approach was first applied to image processing by Golay [6] and has been subsequently applied and extended by Preston and others [13]. Preston [13, Chap. 6] reported specific thinning results using three subfields for hexagonal tessellations and four subfields for rectangular tessellations. Since a smaller number of subfields will tend to produce faster parallel thinning, we are considering the two subfields case illustrated in Figure 4. We have divided the image space into two subfields in a checkerboard pattern. Subfields are activated in the order 1, 2, 1, 2, ... in successive iterations; and thus, only the pixels in one subfield can change state in any iteration. As a consequence only the four diagonal neighbors of each pixel can change state simultaneously with that pixel.

We have adapted the thinning algorithm of Rosenfeld and Kak [17] to this construct:

Algorithm 2 (A2)

A one of S , p is deleted (one changed to zero) iff all of the following conditions are met:

- $C(p) = 1$;
- p is 4-connected to \bar{S} ; and
- $B(p) > 1$.

This deletion rule is applied in parallel to all pixels in a given subfield for a given iteration. Subfields, as described in Figure 4, are alternatively evoked. ($C(p)$ and $B(p)$ are defined in Section 3.) Thinning stops when no further deletions occur.

Condition **a** helps to preserve connectivity and avoids deletion of pixels in the middle of medial curves. Condition **b** guarantees that only boundary pixels are candidates for deletion and condition **c** preserves the

endpoints of medial curves. Thinning proceeds by successively removing boundary pixels until the thinned objects are sufficiently "curve-like" to fail conditions **a** or **c**. $C_s(p)$ in equation (4) can be used to compute condition **a** since condition **b** guarantees that p is not deletable if p has four side ones in its eight-neighborhood.

This algorithm preserves the connectivity properties of the image. Objects of S are neither disconnected nor are originally disjoint objects connected and no object can be completely deleted. The connectivity properties for \bar{S} are also preserved. Proofs for these assertions are given in Appendix B.

5. RESULTS AND DISCUSSION

Two-Subiteration Thinning Algorithms

Algorithms 1 (A1) and 2 (A2) are compared with other two-subiteration algorithms: Zhang and Suen (ZS) [21]; Lü and Wang (LW) [10], and Stefanelli and Rosenfeld (SR) [19]. The relative algorithm performances are compared in two ways:

- We consider whether the requirements of goals 1, 2 and 3(a) (defined in Section 2) are satisfied.
- We simulate parallel operation of the algorithms in Fortran on a Vax 8600 system over several different test sets and measure the following:
 - m_1 , defined as the number of iterations taken to reach the medial curve;
 - m_2 , defined as the percentage of redundant pixels left in the medial curve, i.e.,

$$m_2 = \frac{\text{number of redundant pixels in the medial curve}}{\text{total number of pixels in the medial curve}} \cdot 100\%$$

where a redundant pixel is defined as a pixel in the medial curve, which is not an endpoint, the deletion of which does not disconnect the curve, and the number of redundant pixels is defined as the maximum number of redundant pixels that can be removed simultaneously without disconnecting the medial curve; and

1	2	1	2	1	2	.	.	.
2	1	2	1	2	1	.	.	.
1	2	1	2	1	2	.	.	.
2	1	2	1	2	1	.	.	.
.






















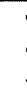














FIGURE 4. Two Subfields Definition Used in Algorithm 2

c. m_3 , defined as the size of the medial curve in number of ones.

The first test set of patterns and corresponding medial curves are given in Table I. (Pixels of the medial curve are denoted by ■ and deleted pixels are denoted with •.) These are typically “difficult” patterns for thinning algorithms and are useful for identifying defects in connectivity and medial curve preservation. Two sets

of results are given for A2 corresponding to the two cases obtained when either subfield is activated first. Where a measure cannot be appropriately applied (e.g., m_2 for the 2×2 or 3×3 cases) an x is entered. We see that our two new approaches, A1 and A2, deal successfully with each pattern while none of the other algorithms can do successful thinning on all of these patterns. The least desirable result for A2 is the overly large result it produces for one of the 3×3 cases.

TABLE I. Thinning Performance for Various Two-Subiteration Thinning Algorithms over “Difficult” Test Patterns

Pattern	Algorithms					
	A1	A2	ZS	LW	SR	
135° Line	 $m_1 = 1$ $m_2 = 0$ $m_3 = 6$	 $m_1 = 1$ $m_2 = 0$ $m_3 = 5$	 $m_1 = 1$ $m_2 = 0$ $m_3 = 6$	 $m_1 = 5$ $m_2 = x$ $m_3 = 1$	 $m_1 = 0$ $m_2 = 45\%$ $m_3 = 11$	 $m_1 = 2$ $m_2 = 44\%$ $m_3 = 9$
45° Line	 $m_1 = 1$ $m_2 = 0$ $m_3 = 7$	 $m_1 = 1$ $m_2 = 0$ $m_3 = 6$	 $m_1 = 1$ $m_2 = 0$ $m_3 = 6$	 $m_1 = 5$ $m_2 = x$ $m_3 = 2$	 $m_1 = 0$ $m_2 = 42\%$ $m_3 = 12$	 $m_1 = 1$ $m_2 = 40\%$ $m_3 = 10$
2 × 2 Square	 $m_1 = 1$ $m_2 = x$ $m_3 = 1$	 $m_1 = 1$ $m_2 = x$ $m_3 = 2$	 $m_1 = 1$ $m_2 = x$ $m_3 = 2$	 $m_1 = 1$ $m_2 = x$ $m_3 = 0$	 $m_1 = 1$ $m_2 = x$ $m_3 = 0$	 $m_1 = 2$ $m_2 = x$ $m_3 = 0$
3 × 3 Square	 $m_1 = 2$ $m_2 = x$ $m_3 = 1$	 $m_1 = 2$ $m_2 = x$ $m_3 = 1$	 $m_1 = 1$ $m_2 = x$ $m_3 = 5$	 $m_1 = 2$ $m_2 = x$ $m_3 = 1$	 $m_1 = 1$ $m_2 = x$ $m_3 = 3$	 $m_1 = 3$ $m_2 = x$ $m_3 = 0$
90° Line	 $m_1 = 1$ $m_2 = 0$ $m_3 = 4$	 $m_1 = 1$ $m_2 = 0$ $m_3 = 5$	 $m_1 = 1$ $m_2 = 0$ $m_3 = 5$	 $m_1 = 1$ $m_2 = 0$ $m_3 = 3$	 $m_1 = 1$ $m_2 = 0$ $m_3 = 3$	 $m_1 = 2$ $m_2 = 0$ $m_3 = 2$
0° Line	 $m_1 = 1$ $m_2 = 0$ $m_3 = 5$	 $m_1 = 1$ $m_2 = 0$ $m_3 = 6$	 $m_1 = 1$ $m_2 = 0$ $m_3 = 6$	 $m_1 = 1$ $m_2 = 0$ $m_3 = 4$	 $m_1 = 1$ $m_2 = 0$ $m_3 = 4$	 $m_1 = 2$ $m_2 = 0$ $m_3 = 3$

ZS, LW, and SR each fail to preserve the 2×2 pattern. ZS and LW fail this test because for example they both remove, in the first iteration, south and east boundary pixels and the north-west corner. ZS fails to preserve endpoints in certain diagonal lines and thus fails to preserve these diagonal medial lines. LW preserves endpoints by increasing the minimum number of one neighbors required to delete a pixel from two to three; but, this results in its failing to thin the two-width diagonal test lines. SR also produces relatively thick diagonal medial lines. A1 and A2 both produce very thin medial lines for diagonal, horizontal and vertical two-width lines. A2 produces a "zigzag" medial line for horizontal and vertical two-width lines (and other orientations not shown).

It is clear from the results in Table I that ZS, LW, and SR do not satisfy thinning goal 1 in that objects in S are not always preserved. We have proven in Appendices A and B that all connectivity properties are maintained by A1 and A2. Goals 2 and 3(a) appear to be reasonably well met by A1, A2, LW, and SR. Only ZS fails badly, on 45° and 135° diagonal patterns of even thickness.

Table II presents quantitative results for the five algorithms over four sets of binary patterns:

1. Set 1 is a set of twelve 25×5 rectangular patterns at orientations of $0, 15, \dots, 165^\circ$.
2. Set 2 is identical to Set 1 except that patterns are of size 40×8 .
3. Set 3 contains three English letters: A, B, and H.
4. Set 4 contains six Chinese characters.

Figures 5, 6, and 7 present typical results for the five algorithms for patterns drawn from these test sets. (Larger dots represent medial curves and smaller dots represent deleted pixels.) The number of iterations, m_1 , the percentage of redundant pixels, m_2 and the medial curve size, m_3 , reported in Table II are the averages over all members of each set. Considering algorithm speed the data for m_1 suggest a ranking from fastest to slowest: A2, LW, ZS, SR, and A1. This ranking holds for

all but Set 1 where LW produces a lower m_1 value. The cases where ZS fails to preserve diagonal lines have especially large m_1 values which are not characteristic of its overall performance and have been excluded from Table II. Considering the thinness of medial curves produced by each algorithm the data for m_2 suggest a ranking from thinnest to least thin: A2, A1, ZS, LW, and SR. A2 produces no redundant pixels in our tests and A1 produces almost no redundant pixels. ZS, LW, and SR each produce substantially more redundant pixels than either A1 or A2. Considering the size of medial curves obtained with each algorithm the data for m_3 suggest a ranking from smallest to the largest: A1, ZS, A2, LW, and SR. A2 produces somewhat larger medial curves than A1 (see m_3 in Table II) because A2 has, as a result of its subfields definition (Figure 4), a tendency to preserve medial curve branches emanating from corners of objects (see Figure 5). For example, if the corner pixel c below,

```

c  d  1
d  1  1
1  1  1

```

is not in the active subfield at the first iteration; then the two d pixels are deleted (since they must be in the active subfield) and on subsequent iterations c is preserved as an endpoint. Thus, A2 tends to produce a medial curve which looks like a medial axis skeleton [16] for many examples, but also produces some unwanted spike branches [17], as on the left side of the letter A in Figure 6. A1 trades off speed with LW and SR to produce substantially thinner and smaller medial curves and to preserve all connectivity properties. It also produces medial curves which are more visually desirable than those produced by A2. A2 achieves the best overall parallel speed and minimizes the number of redundant pixels, but produces a somewhat larger medial curve than A1.

Algorithm A2 achieves its speed advantage over LW principally because of its superior performance on hori-

TABLE II. Thinning Performance for Various Two-Subiteration Thinning Algorithms over Four Pattern Sets

Patterns	Measures	Algorithms				
		A1	A2	ZS	LW	SR
Set 1	m_1	4.33	3.00	3.30*	2.67	3.92
25×5	m_2	0	0	12.1%*	20.3%	24.5%
Rectangles	m_3	18	21	21*	24	23
Set 2	m_1	7.67	4.58	6.30*	5.17	6.75
40×8	m_2	0	0	10.7%*	15.4%	25.7%
Rectangles	m_3	30	37	33*	40	39
Set 3	m_1	8.00	4.67	5.00	5.00	6.33
English	m_2	0.85%	0	6.91%	6.75%	14.1%
Letters	m_3	78	83	82	84	88
Set 4	m_1	9.67	5.67	6.83	6.83	7.67
Chinese	m_2	0.17%	0	6.21%	7.00%	14.5%
Characters	m_3	286	306	304	312	332

* 45° and 135° cases excluded

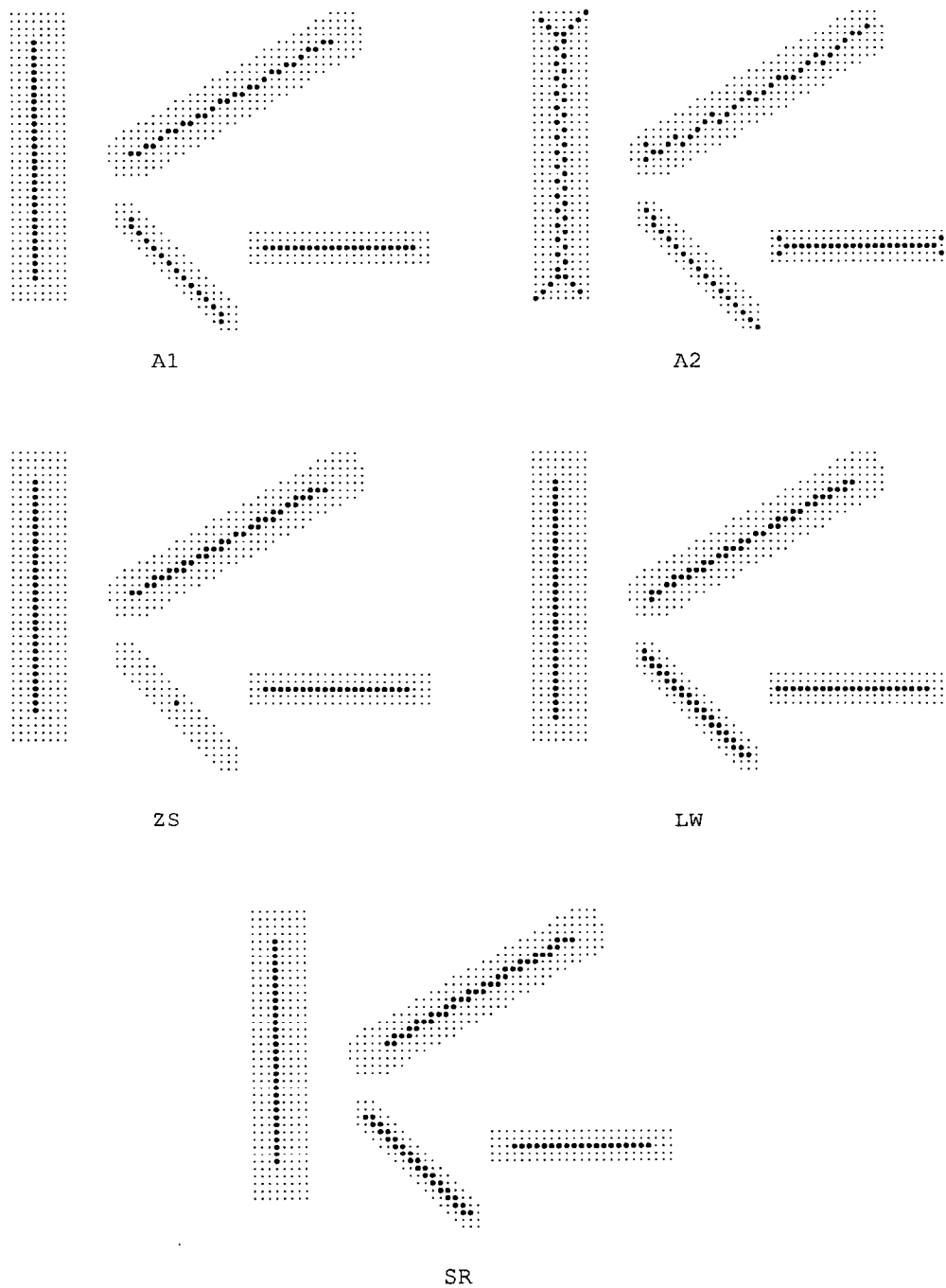


FIGURE 5. Thinning Results for Algorithms A1, A2, ZS, LW, and SR for Lines of Various Orientations Drawn from Test Sets 1 and 2

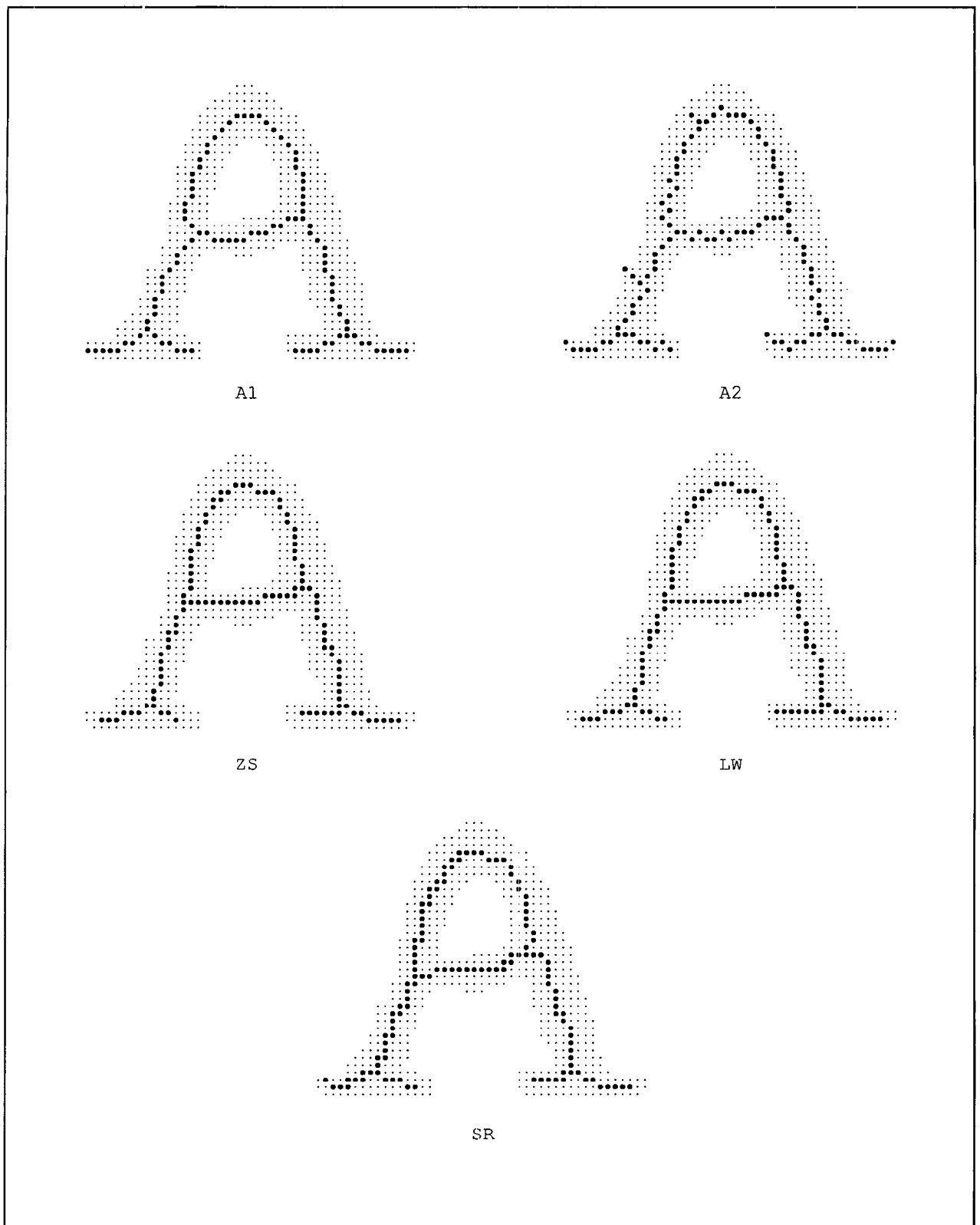


FIGURE 6. Thinning Results for Algorithms A1, A2, ZS, LW, and SR for the English Letter A Drawn from Test Set 3

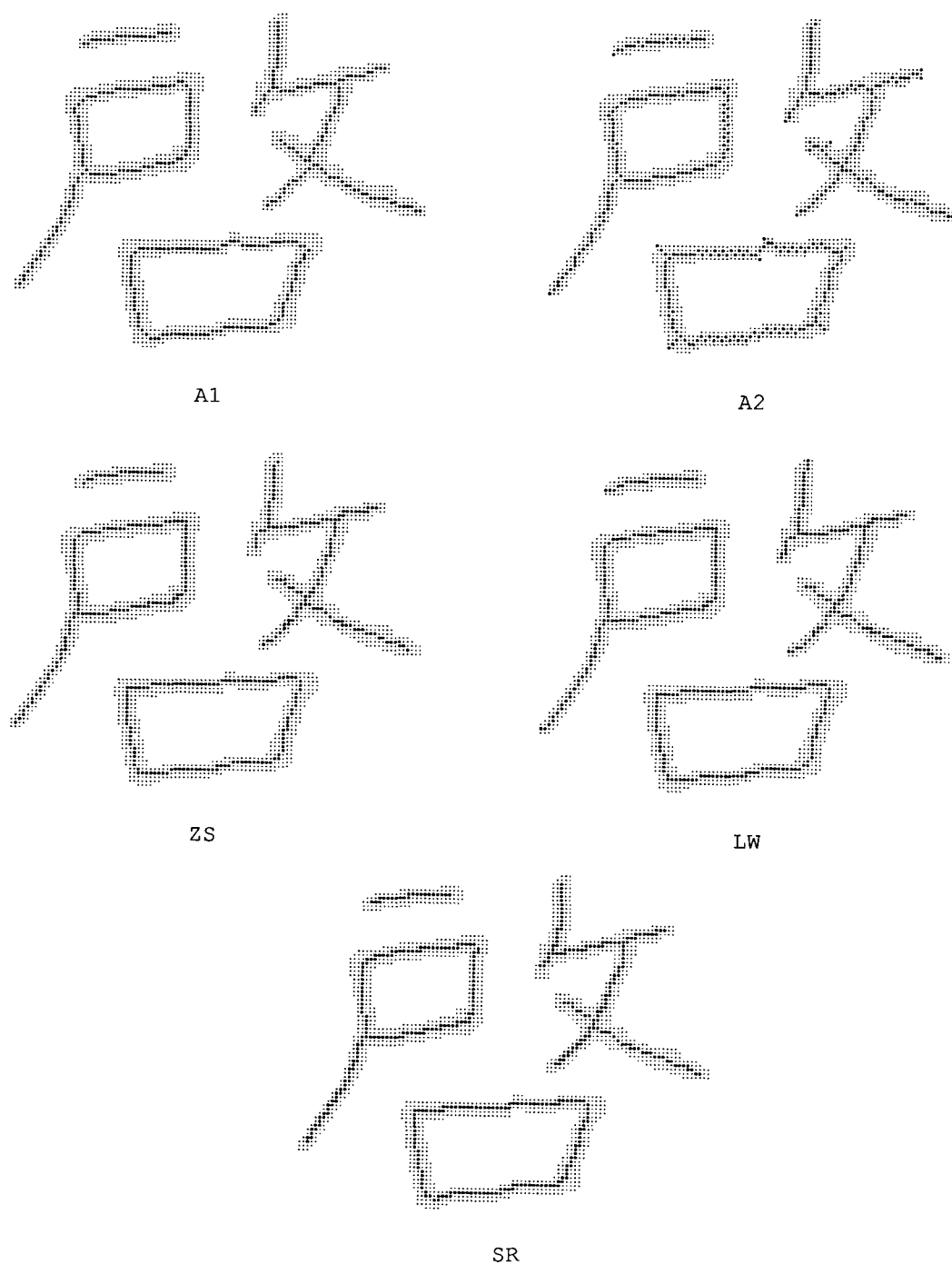


FIGURE 7. Thinning Results for Algorithms A1, A2, ZS, LW, and SR for a Chinese Character Drawn from Test Set 4

zontal and vertical rectangular objects. Since a two-subiteration algorithm like LW cannot delete pixels from north and south or east and west boundaries within one subiteration [15], the best performance achievable on an $m \times n$ pixels rectangle ($m > n$), oriented horizontally or vertically, appears to be $n - 1$ iterations. (Interior pixels cannot be deleted, otherwise holes may be created.) A fully parallel approach which only deletes boundary ones (ones four-connected to 5) would require $\lfloor n/2 \rfloor$ iterations. A2 approaches this fully parallel performance as n grows large since a non-boundary pixel which is four-adjacent to a boundary pixel can be deleted on the iteration following the deletion of the boundary pixel. Figure 8 illustrates this process by indicating the iteration in which each deleted pixel is removed. We note that in the second iteration A2 deletes *all* boundary pixels which are not on the medial curve. This property applies to all iterations after the first and A2 typically requires $\leq \lfloor n/2 \rfloor + 1$ iterations (when $n > 1$) for these $m \times n$ rectangles. A1, ZS, and LW each require four iterations on such rectangles of width five and typically require $n - 1$ iterations for these $m \times n$ rectangles. LW is faster than A2 on diagonal rectangles (45° and 135°) but this speed advantage is principally due to LW's failure to produce thin diagonal medial lines. A diagonally ori-

1	2	1	2	1	2	1	2	1	2	1	2	1	2	1
2	*	2	3	2	3	2	3	2	3	2	3	2	*	2
1	2	*	*	*	*	*	*	*	*	*	*	*	2	1
2	*	2	3	2	3	2	3	2	3	2	3	2	*	2
1	2	1	2	1	2	1	2	1	2	1	2	1	2	1

FIGURE 8. Iterations when A2 Deletes Pixels for a 5×15 Horizontal Rectangle (Iteration 1 Locations Indicate the Position of Subfield 1; Medial Curve Denoted by *)

ented boundary (45° or 135°) is composed of pixels all within a single subfield and is entirely deletable by A2 in one iteration when that subfield is active. In the iteration after these boundary pixels are deleted, the new boundary pixels are in the opposite subfield which is currently active and are in turn entirely deletable. As a result successive iterations of A2 are able to delete all of these diagonal boundary pixels until the region of the medial curve is reached. Thus, the parallel speed of A2 (like LW) on these diagonal boundaries comes close to that achievable by a fully parallel approach. Finally, results on the natural characters, Sets 3 and 4, indicate that A2 tends to produce faster thinning for naturally occurring distributions of line orientations.

A2 produces medial curves with a "zigzag" pattern for certain object orientations and widths (e.g., horizontal and vertical rectangles of even width). Although visually disconcerting these "zigzag" patterns are of

similar complexity to normal straight medial curves in a chain code representation and require no more storage. But, the "zigzag" patterns may represent a disadvantage if an interpixel distance of $2^{1/2}$ between two diagonally adjacent pixels is used when estimating curve length. Although this is an unusual result compared to traditional thinning algorithms, this form of medial line can provide, for example, the least biased estimate of the position of the central axis of a long two-width rectangle. Corner configurations for medial curves of rectangular objects of 0° or 90° orientation are symmetrical for rectangles with odd lengths and widths, since this places all corners in the same subfield. For other cases where all corners are not in the same subfield, corner configurations are not precisely symmetrical. (See Figures 5 and 8 for examples.)

Fully Parallel Thinning Algorithms

Two results [4, 9] which appeared after the submission of our work are pertinent and we take the opportunity to discuss these in our revision. Holt et al. [9] have proposed a fully parallel thinner which uses an effective support of size 4×4 . In order to fairly compare the parallel speed of competing algorithms we should restrict supports for each in the same way. If we apply the 3×3 support restriction to [9], this algorithm is comparable to other two-subiteration algorithms and has been shown to exhibit substantially lower parallel speed than LW [7]. The later work of Chin et al. (CWSI) also proposes a fully parallel thinner which uses 3×3 operators for reduction and 1×4 operators for restoration; thus, the overall operator support is over a 4×4 region [4]. Again a comparison with algorithms using operators restricted to 3×3 support is tenuous. If CWSI is used with a 3×3 support restriction, then it can be configured as a two-subiteration algorithm where each fully parallel iteration over a 4×4 support requires two iterations over a 3×3 support. We have evaluated CWSI over our test sets and report the results in Table III where iteration counts are for the fully parallel 4×4 support. Figure 9 shows typical results for CWSI for the test objects of Figures 5, 6, and 7. The new fully parallel approach has higher iteration counts than LW and A2 for all test sets. If we apply the 3×3 support restriction to CWSI, the iteration counts double and CWSI compares even less favorably to LW and A2. CWSI produces fewer redundant pixels (as measured by m_2) than LW, but considerably more than A1 or A2. For example, CWSI fails to delete any pixels in any two-width 135° line of the following sort

```

11
 11 .
 11

```

But, CWSI does have on average rather small medial curve sizes (m_3) approaching those achieved by A1. The principal reason for identifying fully parallel ap-

TABLE III. Thinning Performance for the Fully Parallel Thinner CWSI [4]

Patterns	Measures		
	m_1	m_2	m_3
Set 1 25 × 5 Rectangles	3.17	6.2%	20
Set 2 40 × 8 Rectangles	5.50	6.8%	32
Set 3 English Letters	5.67	2.2%	77
Set 4 Chinese Characters	8.00	2.7%	291

proaches is to reduce number of required iterations, but CWSI does not appear to achieve that objective when compared to LW and A2. This unexpected result is partly explained if we consider the 30° line case illustrated in Figure 10 where A2 outperforms CWSI. Here pixels, p , with neighborhoods of the following form (or 90° rotations)

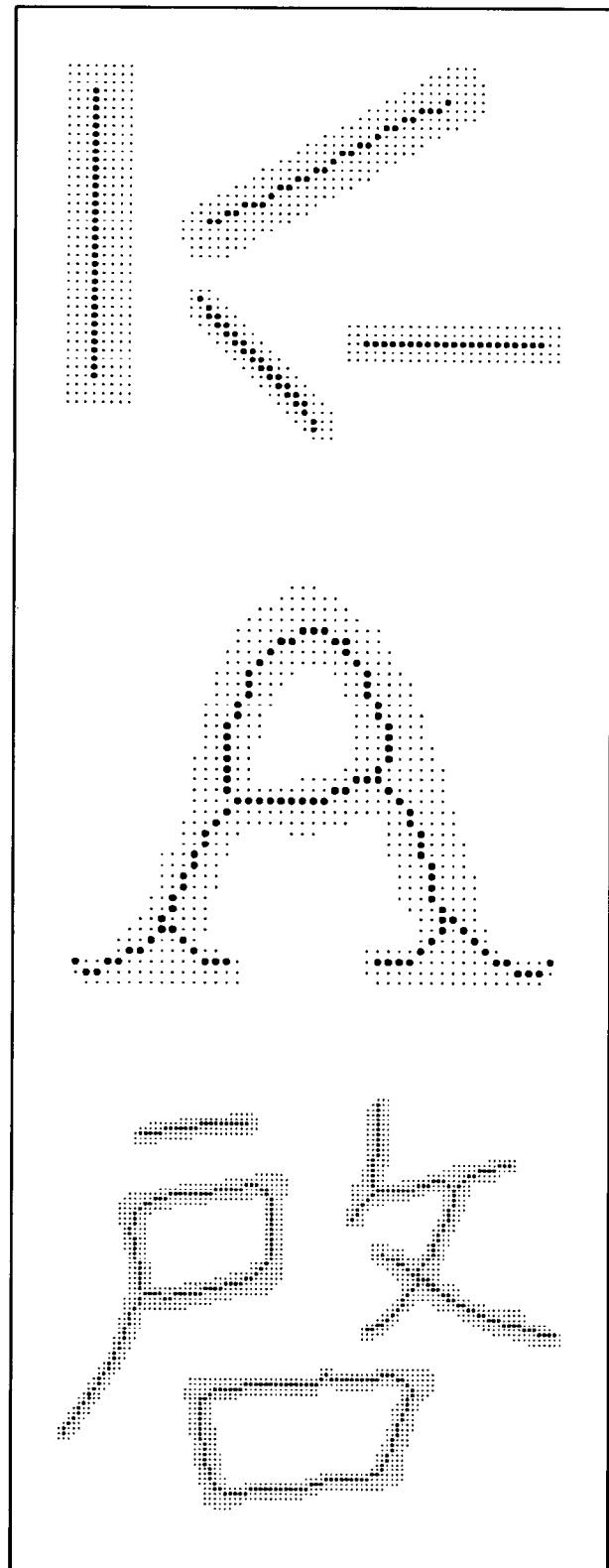
$$\begin{array}{ccc} 1 & 1 & 1 \\ 1 & p & 1 \\ 1 & 0 & 0 \end{array}$$

are not deletable by CWSI, but are deleted by A2 when p is in the active subfield.

6. SUMMARY

Two parallel thinning algorithms have been presented which use two subiterations. The first, A1, modifies the algorithms of Zhang and Suen [21] and Lü and Wang [10] to preserve connectivity and produce thin medial curves. The second, A2, uses two subfields with a modification of the classical thinning approach of Rosenfeld and Kak [17]. Both new algorithms preserve image connectivities and produce thinner results than other two-subiteration algorithms considered in our tests [10, 19, 21]. Algorithm A1 provides the smallest medial curves with thinness comparable to Algorithm A2. A2 produces the highest parallel speed among two-subiteration thinners and its parallel speed is also shown to be superior to a recently reported fully parallel thinning approach [4].

Acknowledgment. The authors acknowledge the thoughtful reviews of the referees which helped to improve the clarity of this manuscript.

**FIGURE 9. Thinning Results for CWSI for the Test Objects of Figures 5, 6, and 7**

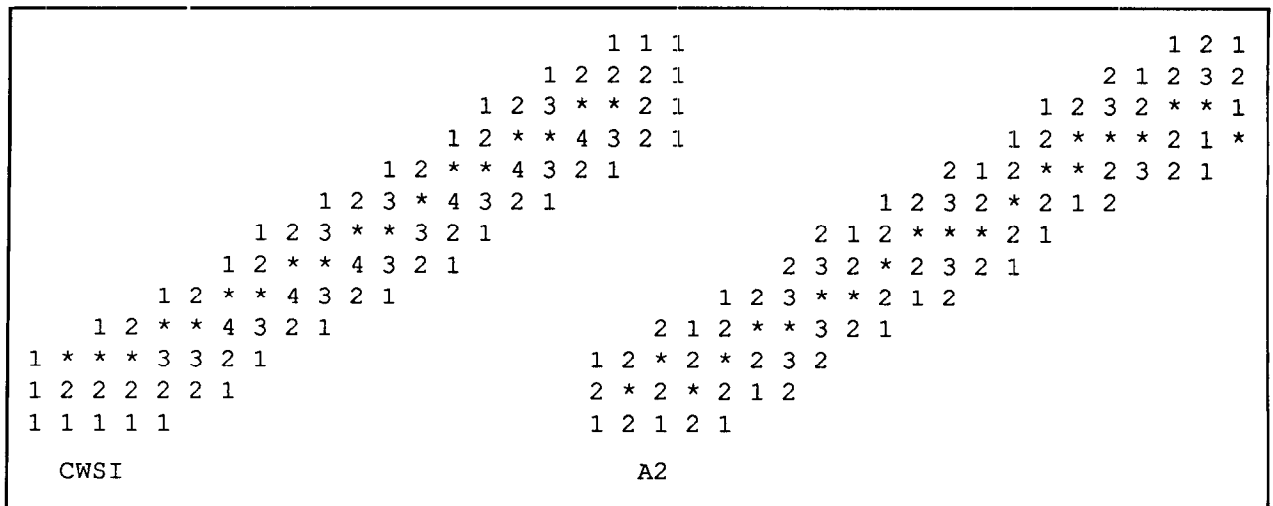


FIGURE 10. Thinning Results on a 30° Line from Set 1 for CWSI and A2 (Numbers Give the Iteration when Pixels Are Deleted and the Medial Curve Is Denoted with *; Iteration 1 Locations Indicate the Position of Subfield 1 for A2)

APPENDIX A

Algorithm 1 preserves the connectivity properties of any binary image as described in the following propositions. Conditions **a**, **b**, **c(1)**, and **c(2)** refer to the required conditions for deletion given for Algorithm 1.

Proposition A1. No object in S can be completely deleted by Algorithm 1 in any iteration.

Proof: We prove the proposition for odd iterations, even iterations follow similarly. Assume the contrary hypothesis; thus, an object, Q , exists which is completely removed in one iteration. Pixels in Q satisfy either of the conditions shown in Figure 3, (a) or (b). Figure 11 repeats these conditions for the neighborhood of a pixel x_1 with specific names for its neighbors. Assume there is at least one pixel, x_1 , in Q satisfying (b) as illustrated in Figure 11(b). Pixel $f \neq 1$ since its neighborhood does not satisfy (a) or (b). Since $f = 0$, then $c = d = e = 0$ to obtain $C(x_1) = 1$; but then $N(x_1) = 1$ and x_1 cannot be deleted. Thus, all pixels in Q must satisfy (a). Figure 11(a) illustrates the neighborhood of one such pixel. Pixel $d \neq 1$ since its neighborhood does not satisfy (a). Since $d = 0$, to achieve $N(x_1) > 1$ while preserving $C(x_1) = 1$, $a = b = c = 0$ and $e = f = g = 1$ or $a = b = c = 1$ and $e = f = g = 0$. In either case two ones (b and c or e and f) will not satisfy (a) and will not be deleted. Thus, Q does not exist and the contrary hypothesis is contradicted. \square

Since Algorithm 1 converts only ones to zeros and only creates new zeros which are four-connected to existing zeros, the following is obvious.

Corollary A.2. Algorithm 1 does not connect any originally disjoint objects of S , nor does it disconnect an existing hole or create any new holes in S .

Proposition A.3. Algorithm 1 does not disconnect any object in S .

Proof: It is sufficient to show that for any eight-connected path $\{x_0, x_1, \dots, x_n\}$ lying within a given object, Q , of S , where only x_1, x_2, \dots, x_{n-1} are deleted at iteration i , there exists an alternate eight-connected path, P , in Q which connects x_0 and x_n and is composed of pixels which are not deleted at i [15]. If we can show this result for $n = 2$, it is straightforward to induce the general result. Thus, we consider the three pixels x_0, x_1 , and x_2 and specifically the eight-neighborhood of x_1 which contains x_0 and x_2 (if Q contains less than three ones, condition **b** will always preserve it). Pixel x_1 is deleted at i and we need to establish an alternate path, P , to maintain connectivity between x_0 and x_2 for any combination of x_0, x_2 positions. P must contain no pixels which can be deleted at i . We give the proof for odd iterations. The proof for even iterations follows similarly. Condition **c(1)** provides two neighborhood conditions where x_1 could be deleted (Figure 11) and we formulate our enumeration of all possible cases around these two conditions. We need not consider cases where x_0 and x_2 are eight-adjacent already. Further, for the case of Figure 11(a) a and g cannot be both ones since otherwise we would have either $N(x_1) = 4$ (if $b = d = f = 1$) or $C(x_1) > 1$ (if b, d or $f = 0$), violating the assumption that x_1 is deleted. Figure 12 illustrates

c	b	a	c	0	0
d	x_1	0	d	x_1	h
e	f	g	e	f	1
(a)			(b)		

FIGURE 11. Neighborhood Conditions Required for Deletion of x_1 in Odd Iterations of A1

the three resulting cases from which the enumerations are taken. For the three cases (a), (b) and (c) there are 16, 16, and 10 possibilities to consider, respectively. Figure 13 gives examples of each of the three cases. For the 16 possibilities derived from Figure 12(a), the pixels (b or d or both) eight-connecting x_0 to x_2 must be ones so that $C(x_1) = 1$. (For the example of Figure 13(a), $b = d = 1$.) These b and d pixels are not deletable as they each fail condition $c(1)$ and thus compose the required alternate path P . The same form of argument applies to each of the possibilities derived from Figure 12(b) and 12(c) except that the pixels d or f or both are required to be ones and compose the path P . (e.g., in the examples of Figure 13(b) and 13(c) $d = f = 1$ and d and f are not deletable since they fail condition $c(1)$). \square

Lemma A.4. For any object, Q , with a hole, H , Algorithm 1 will preserve after any iteration at least two ones which are not eight-adjacent.

Proof: Our proof is given for odd iterations. The proof for even iterations is similar. Referring to Figure 14 let p_1 be the one to the immediate left of a leftmost zero, z_1 , in the hole H , and p_2 be the one to the immediate right of a rightmost zero, z_r . (z_1 and z_r could be the same zero.) By the definition of p_1 , neither $\{a_2, a_3\}$ nor $\{b_2, b_3\}$ can contain all zeros. Similarly from the definition of p_2 , neither $\{c_2, c_3\}$ nor $\{d_2, d_3\}$ can contain all zeros. Now the claim is that after any odd iteration: (1) at least one of the two elements q_1 and p_1 will be preserved, and (2) p_2 will be preserved. To show (1), first suppose $q_1 = 0$. Then p_1 is not deleted since $C(p_1) \neq 1$. Now assume $q_1 = 1$. If q_1 is not deletable, we are done. If q_1 is deletable, it must satisfy Figure 3(b) and $a_1 = a_2 = 0$ and $b_2 = 1$. Since $a_2 = 0$, $a_3 = 1$ and $C(p_1) \neq 1$ and p_1 is not deleted. To show (2) first assume $q_2 = 0$. Then p_2 cannot be deleted since $C(p_2) \neq 1$. Next let $q_2 = 1$ and assume p_2 is deletable. To satisfy Figure 3(b) $c_2 = c_1 = 0$ and $d_1 = 1$. But since $c_2 = 0$ and $c_3 = 1$, we have $C(p_2) \neq 1$. This contradicts the assumption that p_2 is deletable; therefore, p_2 is preserved. Finally, p_2 is not adjacent to p_1 or q_1 for any H . \square

Proposition A.5. Algorithm 1 does not connect any hole of S to another distinct hole or the background of S .

Proof: Let H be any hole in S . Originally an object, Q , surrounds H . Assume H becomes four-connected to the background or another hole, via a path W of zeros, for the first time at iteration i . Then for any two non-

c	b	a	c	b	0	c	0	0
d	x_1	0	d	x_1	0	d	x_1	h
e	f	0	e	f	g	e	f	1
(a)			(b)			(c)		

FIGURE 12. Classes of Neighborhoods of x_1 from which Enumerations are Formed in the Proof of Proposition A.3

c	b	x_0	c	x_0	0	x_0	0	0
d	x_1	0	d	x_1	0	d	x_1	x_2
e	x_2	0	e	f	x_2	e	f	1
(a)			(b)			(c)		

FIGURE 13. Example Neighborhoods from Each Class Shown in Figure 12

a_1	a_2	a_3		c_3	c_2	c_1
q_1	p_1	z_1	H	z_r	p_2	q_2
b_1	b_2	b_3		d_3	d_2	d_1

FIGURE 14. Conditions around a Hole, H , Used in the Proof of Lemma A.4

adjacent pixels, x_1 and x_2 , remaining in Q after iteration i , there was an eight-connected path, R , connecting x_1 and x_2 in Q at $i - 1$ which was disconnected by W at iteration i . Lemma A.4 guarantees that Q contains at least two ones after i , which are not eight-adjacent; thus x_1 and x_2 exist. But the proof of Proposition A.3 shows that for path R an alternate path, P , exists at i connecting x_1 and x_2 which is in the neighborhood of pixels of R and thus belongs to Q . P would disconnect the hypothetical path W and thus W cannot exist and holes are preserved.

Proposition A.4. $C(p) = C_s(p)$ (4) for all neighborhoods of p except for the case when p has four side ones; and in this case $C_s(p) = 0$ and $C(p) = 1$.

Proof: Both operations are symmetrical with respect to 90° rotations; thus, we need to show the result for

only one orientation. We enumerate the cases we need consider over the number of p 's side ones:

x	1	x	x	1	x	a	0	x
1	p	1	0	p	1	0	p	1
x	1	x	x	1	x	x	1	x
(a)			(b)			(c)		
x	1	x	a	0	x	a	0	b
0	p	0	0	p	1	0	p	0
x	1	x	b	0	x	c	0	d
(d)			(e)			(f)		

where x refers to a "don't care" pixel (either a zero or one) and pixels a , b , c , and d are referenced below. When p has exactly four side ones, case (a). $C(p) = 1$ and $C_s(p) = 0$. When p has exactly three side ones, case (b), $C(p) = 1$ is guaranteed and $C_s(p) = 1$. There are 2 cases to consider when p has exactly two side ones, cases (c) and (d). In case (c) $C(p) = C_s(p) = 1$ when $a = 0$ and $C(p) = C_s(p) = 2$ when $a = 1$. For case (d) $C(p) = C_s(p) = 2$. For case (e) $C(p) = C_s(p) = 1 + a + b$ and similarly for case (f) $C(p) = C_s(p) = a + b + c + d$. □

APPENDIX B

Algorithm 2 preserves the connectivity properties of any binary image as described in the following propositions. Conditions **a**, **b**, and **c** refer to the three required conditions for deletion given for Algorithm 2.

Proposition B.1. No object in S can be completely deleted by Algorithm 2 in any iteration.

Proof: Assume the contrary hypothesis. Thus, an object, Q , exists which is deleted in one iteration and Q must lie in only one subfield. Consider the neighborhood of any pixel, p , in Q . This neighborhood will contain from 0 to 4 ones in the subfield of p . Neighborhoods with only 0 or 1 ones fail condition **c**. Neighborhoods with 2, 3, or 4 ones cannot be eight-connected, by the subfield definitions; and thus, $C(p) > 1$ failing condition **a**. Thus, p cannot be deleted which contradicts the contrary hypothesis. □

Since Algorithm 2 converts only ones to zeros and only creates new zeros which are four-connected to existing zeros, (condition **b**) the following is obvious.

Corollary B.2. Algorithm 2 does not connect any originally disjoint objects in S , nor does it disconnect an existing hole or create any new holes in S .

Proposition B.3. Algorithm 2 does not disconnect any object in S .

Proof: We use the approach in the proof of Proposition A.3. Thus, it is sufficient to consider an object, Q , with three or more ones and show that given any three ones x_0 , x_1 , and x_2 forming a path in Q where x_0 and x_2 are not eight-adjacent; deletion of x_1 at any iteration i will guarantee that an alternative path, P , exists which connects x_0 to x_2 . Pixels of P must not be deletable during iteration i . Refer to x_1 's subfield as S_1 and the other subfield as S_2 . Since pixels in S_2 will not change during i , it is sufficient to show that P exists and is wholly in S_2 for any possible positions of x_0 and x_2 in the neighborhood of x_1 with x_0 not eight-adjacent to x_2 . One such case is illustrated below

x_0	a
d	x_1 b
c	x_2

In this case x_1 must satisfy condition **a** to be deletable and this requires $a = b = 1$ or $d = c = 1$. In either case P exists drawn from the set of pixels $\{a, b, c, d\}$ which belong wholly to S_2 . All cases lead to a similar result where P must exist to make $C(x_1) = 1$ and is drawn from the set $\{a, b, c, d\}$. □

Proposition B.4. Algorithm 2 does not connect any hole of S to another distinct hole or the background of S .

Proof: Assume the contrary hypothesis holds and that two originally unconnected components of \bar{S} become four-connected for the first time at iteration i via path W . W contained a set of one or more ones, T , which were deleted at i . Since no two members of either subfield are four-connected through pixels in the same subfield, none of the members of T are four-adjacent. Thus, in the path W any member of T , x_1 , is surrounded by two pixels, x_0 and x_2 , which were zero before i . (We assume that W is defined to include at each end zeros belonging to the distinct components of \bar{S} .) For at least one member of T , x_1 , x_0 , and x_2 were not four-connected before i , but are four-connected by x_1 at i . Otherwise, the two distinct components of \bar{S} were already four-connected before i . We show in the following that this case cannot occur. Consider the neighborhood of x_1 . x_0 and x_2 belong to this neighbor-

hood, are four-connected to x_1 and partition the other six neighbors into two four-connected sets as illustrated for one case below

a	b	c
x_0	x_1	x_2
d	e	f

To be deleted x_1 must satisfy conditions **a** and **c**, but since $x_0 = x_2 = 0$ this can only occur if exactly one of the neighbor sets contains all zeros (e.g., $a = b = c = 0$ or $d = e = f = 0$ in the example). All cases produce this form of result where for x_1 to be deletable x_0 and x_2 are already four-connected, contradicting the contrary hypothesis. □

REFERENCES

1. Arcelli, C., and Di Baja, G.S. A width-independent fast thinning algorithm. *IEEE Trans. Patt. Anal. and Mach. Intell. PAMI-7*, 4 (July 1985), 463-474.
2. Arcelli, C., Cordella, L., and Levialdi, S. Parallel thinning of binary pictures. *Electronics Letters* 11, 7 (Apr. 1975), 148-149.
3. Arcelli, C., Cordella, L.P., and Levialdi, S. From local maxima to connected skeleton. *IEEE Trans. Patt. Anal. and Mach. Intell. PAMI-3*, 2 (Mar. 1981), 134-143.
4. Chin, R.T., Wan, H-K., Stover, D.L., and Iverson, R.D. A one-pass thinning algorithm and its parallel implementation. *Comp. Vis. Graphics Image Process* 40 (1987), 30-40.
5. Goetcherian, V. From binary to grey tone image processing using fuzzy logic concepts. *Patt. Recognition* 12 (1980), 7-15.
6. Golay, M.J.E. Hexagonal parallel pattern transformations. *IEEE Trans. on Computers C-18*, 8 (Aug. 1969), 733-740.
7. Hall, R.W. Fast parallel thinning algorithms: Parallel speed and connectivity preservation. *Commun. ACM* 32, 1 (Jan. 1989), 124-131.
8. Hilditch, C.J. Linear skeletons from square cupboards. In *Machine Intelligence* 4, B. Meltzer and D. Michie, Eds. American Elsevier, New York, 1969, 403-420.
9. Holt, C.M., Stewart, A., Clint, M., and Perrott, R.H. An improved parallel thinning algorithm. *Commun. ACM* 30, 2 (Feb. 1987), 156-160.
10. Lü, H.E., and Wang, P.S.P. A comment on A fast parallel algorithm for thinning digital patterns. *Commun. ACM* 29, 3 (Mar. 1986), 239-242.
11. Pavlidis, T. A flexible parallel thinning algorithm. In *Proceedings of the Conference on Pattern Recognition and Image Processing* (Dallas, Texas, Aug. 3-5, 1981), IEEE, New York, 1981, pp. 162-167.
12. Pavlidis, T. *Algorithms for Graphics and Image Processing*. Springer-Verlag, Berlin, 1982.
13. Preston, K., and Duff, M.J.B. *Modern Cellular Automata*. Plenum, New York, 1984.
14. Rosenfeld, A. Connectivity in digital pictures. *JACM* 17, 1 (Jan. 1970), 146-160.
15. Rosenfeld, A. A characterization of parallel thinning algorithms. *Information and Control* 29 (1975), 286-291.
16. Rosenfeld, A., and Kak, A. *Digital Picture Processing*. Academic Press, New York, 1976.
17. Rosenfeld, A., and Kak, A. *Digital Picture Processing Vol. 2*. Academic Press, New York, 1982.
18. Rutovitz, D. Pattern recognition. *J. Royal Statist. Soc.* 129, (1966), 504-530.
19. Stefaneli, R., and Rosenfeld, A. Some parallel thinning algorithms for digital pictures. *JACM* 18, 2 (Apr. 1971), 255-264.

20. Tamura, H. A comparison of line thinning algorithms from digital geometry viewpoint. In *Proceedings of the International Conference on Pattern Recognition* (Kyoto, Japan, Nov. 7-10, 1978). IEEE, New York, 1979, pp. 715-719.
21. Zhang, T.Y., and Suen, C.Y. A fast thinning algorithm for thinning digital patterns. *Commun. ACM* 27, 3 (Mar. 1984), 236-239.

CR Categories and Subject Descriptors: I.4 [Image Processing]: I.5.2 [Pattern Recognition]: Design Methodology—*pattern analysis*; I.5.4 [Pattern Recognition]: Applications—*computer vision*

General Terms: Algorithms, Performance, Theory

Additional Key Words and Phrases: Parallel algorithms, preservation of connectivity, thinning binary patterns

ABOUT THE AUTHORS:

ZICHENG GUO is a research assistant working toward his Ph.D. in the Department of Electrical Engineering at the University of Pittsburgh. His current research interests are in parallel algorithms and architectures, and their applications in image processing and pattern recognition. Author's Present Address: Department of Electrical Engineering, 348 Benedum Engineering Hall, University of Pittsburgh, Pittsburgh, PA 15261.

RICHARD W. HALL is an associate professor of electrical engineering at the University of Pittsburgh. His current research interests are in studies of parallel algorithms and architectures for visual information processing. Author's Present Address: Department of Electrical Engineering, 348 Benedum Engineering Hall, University of Pittsburgh, Pittsburgh, PA 15261.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.