# Novel method for analysis of printed circuit images

by Jon R. Mandeville

**To keep pace with the trend towards increased circuit integration, printed circuit patterns are becoming denser and more complex. A variety of automated visual inspection methods to detect circuit defects during manufacturing have been proposed. This paper describes a method which is a synthesis of the *reference-comparison* and the *generic-property* approaches that exploits their respective strengths and overcomes their respective weaknesses. It is based on the observation that the local geometric and global topological correctness of a printed circuit can be inferred from the correctness of simplified, skeletal versions of the circuit in a test image. These operations can be realized using simple *processing elements* which are well suited for implementation in hardware.**

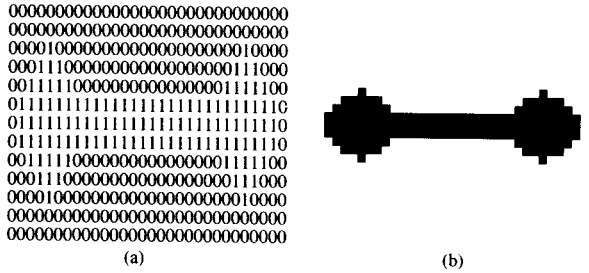
## 1. Introduction

Electronic packaging technology is evolving towards interconnecting more integrated circuits on a single printed circuit board or substrate. As a result, printed circuit boards and multilayer ceramics are increasing in size and they contain more layers. In addition, the printed circuits themselves are becoming smaller and more complex. For example, for use in its high end computers like the 3081, IBM now manufactures printed circuit boards (called TCM boards) as large as 600 by 700 mm (24 by 28 in.), composed of up to 20 layers, and with circuit features as small as 0.081 mm (3.5 mils) [1]. IBM also routinely manufactures 90-mm (3.5-in.) multilayer ceramic substrates consisting of up to 33 layers and using 0.12-mm (5-mil) circuit features [2].

As these packaging technologies become increasingly complex, substrates become more costly not only to produce but also to replace in the field. Therefore, it is important that quality control methods keep pace with the trend towards larger circuit areas, more complex circuits, and smaller circuit features. In particular, the printed circuits on each layer must be carefully inspected before that layer is used to make up a composite of layers. Although electrical testing can detect some defect types, only human visual (or automated optical) inspection can reliably detect many of the "fatal" defects.

Unfortunately, human visual inspection is labor intensive and therefore costly. In addition, human subjectivity and the tedium of the work contribute to variations in the quality of inspection. Interest in automating the visual inspection process is motivated by the desire to reduce labor costs and standardize inspection quality.

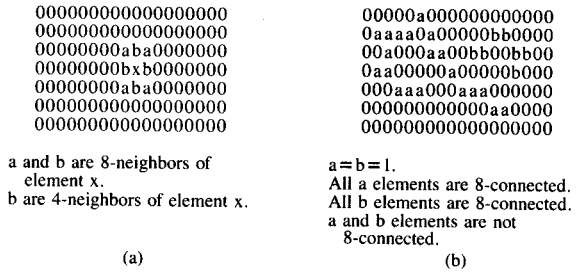
A variety of approaches for automated optical inspection of printed circuits have been reported over the last decade; see the surveys of Chin [3] and Kruger and Thompson [4]. These approaches typically use an analog subsystem for part handling and image acquisition and a digital subsystem for image analysis and overall system control. Such systems usually do not analyze grey scale analog images; instead most methods are based on the analysis of *discrete, binary images* generated by sampling analog images on a regular grid and thresholding the result to a zero or one. Discrete, binary images (or just images) are $n$ by $m$ matrices whose




(a) (b)

**Figure 1**

Discrete binary image: (a) matrix representation; (b) picture representation (black and white square pixels correspond to ones and zeros, respectively, in the array).




a and b are 8-neighbors of element x.
b are 4-neighbors of element x.

(a)

a=b=1.
All a elements are 8-connected.
All b elements are 8-connected.
a and b elements are not 8-connected.

(b)

**Figure 2**

Neighbors and connectivity: (a) 8- and 4-neighbors of arbitrary element x in the matrix; (b) example of 8-connected ones in a binary array.

elements (*pixels*) are zero or one. An image of an idealized printed circuit *trace* ending in *pads* is shown in **Figure 1.** (In a printed circuit board, traces are analogous to wires and pads to terminals.)

Most proposed methods for the analysis of printed circuit images are variations of either the *reference-comparison* or the *generic-property* approaches. In general, the reference-comparison approach uses complete knowledge of the circuit under test, whereas the generic-property approach uses knowledge of properties common to a circuit family but not knowledge of the specific circuit under test. There are two types of reference comparison. The simpler approaches involve some kind of direct image comparison, e.g., boolean exclusive OR between pixels in a test image and pixels in an idealized reference image. Somewhat more sophisticated approaches involve recognition of circuit features in the test image (e.g., pads, corners, etc.) followed by comparison against a reference. The generic-property approach also takes two forms. One is based on the notion that idealized circuit features are simple, regular geometric shapes, whereas defects typically are not. With this approach, one looks for unexpected irregular features. The second approach is based on directly verifying *design rules*, e.g., trace width, feature spacing, pad location and size, etc. In both forms, defects are usually detected using strictly local neighborhood processing throughout the test image.

In Section 2 of this paper we describe a generic method for the analysis of printed circuit images that is a synthesis of the reference-comparison and the generic-property approaches. This method is a powerful and flexible analysis technique that can be used to detect typical circuit defects. It replaces both image comparison and design rule checkers, exploiting their strengths and overcoming their weaknesses. In Section 3 we describe algorithms based on the generic method for verifying trace width, feature spacing, and pads; we also present results of detecting simulated defects using these algorithms. In Section 4 we briefly discuss the implementation of the basic image processing operations in software and on special purpose digital hardware.

## 2. Overview of analysis method

• *Definitions*

The concepts underlying our method are derived from recent work in *discrete geometry* on the geometric description and analysis of discrete, binary images [5–8]. Specifically, the concepts of *neighbors*, *connectivity*, and *entity* formalize the intuitive notion of what distinct objects are contained in an image. In addition, the image-to-image transformations, *contraction*, *expansion*, and *thinning*, provide a formalism from which to infer the shape, size, and topology of entities [9–11]. Below we define those concepts necessary for an intuitive understanding of the method and the inspection algorithms described later. See the Appendix for a complete description of the image processing operations.

*Neighbors*

An element $I(i, j)$ of a binary image is an *8-neighbor* of another element $I(m, n)$ if $\max [|i - m|, |j - n|] \leq 1$. The *8-neighborhood* of an element is a set containing the element and its 8-neighbors. An image element $I(i, j)$ is a *4-neighbor* of another element $I(m, n)$ if $|i - m| + |j - n| \leq 1$. See **Figure 2(a).**

*Connectivity*

Two nonzero elements are *connected* if there exists an unbroken sequence of nonzero 8-neighbors between the two elements. See **Figure 2(b).**

*Entity*

An *entity* is a connected set of ones in an image. Examples of entities typically found in printed circuits are depicted in **Figure 3(a).** An entity may be a single point, a connected sequence, or a blob of arbitrary shape; it may have holes, be as large as the entire field of view, etc.

*Expansion (contraction)*

We define expansion explicitly here. Contraction is realized by first expanding the complement of an image and then taking the complement of the result. The operation *4-expansion* sets all zeros in an image to one if they have a 4-neighbor equal to one; *8-expansion* sets all zeros in an image to one if they have an 8-neighbor equal to one. **Figure 4** illustrates the result of 8-expanding (8-contracting) the entities in Fig. 3(a) twice. Note that expansion fills in detail: if carried far enough, entities merge. Contraction shrinks entities: if carried far enough, entities break up, e.g., entity B, or disappear altogether, e.g., entity A. *Oct-expansion* alternates 4-expansion with 8-expansion. This results in a better discrete approximation to circular expansion than 4- or 8-expansion used alone; see **Figure 5.**

*Thinning*

In general, *thinning* reduces an entity to its *skeleton*, a simplified version contained in the original entity that

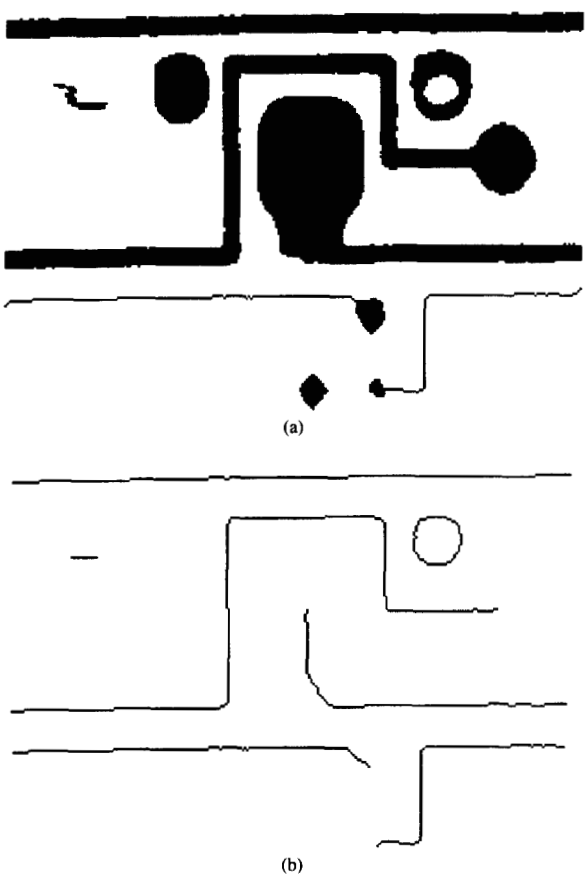



**Figure 3**

Discrete, binary image of typical printed circuit entities: (a) typical entities; (b) their skeletons.

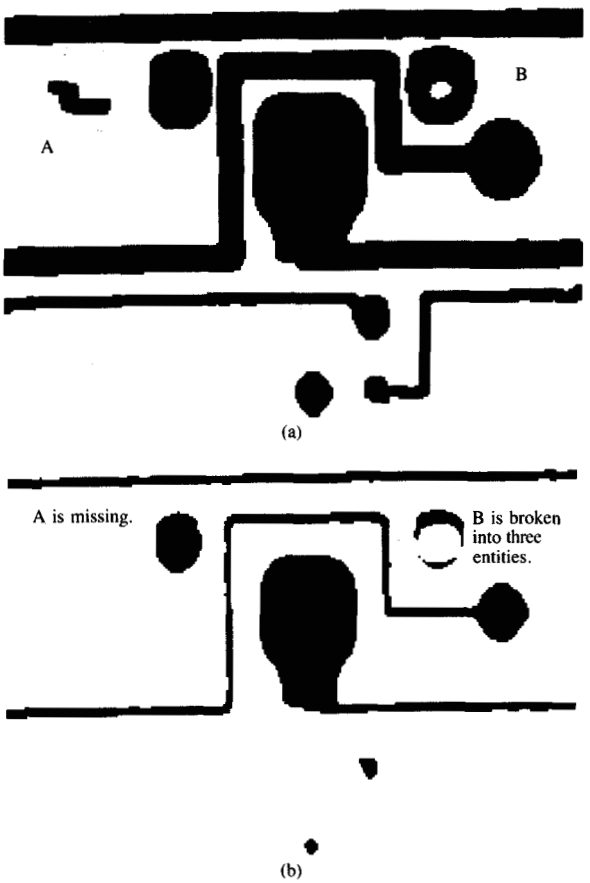



**Figure 4**

Entities from Fig. 3(a) which have been (a) expanded twice and (b) contracted twice.

```
8 8 8 8 8 8 8 8 8
  8 8 7 7 7 7 7 7 8 8
    8 8 7 6 6 6 6 6 7 8 8
      8 8 7 6 6 5 5 5 5 6 7 8 8
        8 8 7 6 6 5 4 4 4 4 5 6 6 7 8 8
          8 7 6 6 5 4 4 3 3 3 4 4 5 6 6 7 8
            8 7 6 5 4 4 3 2 2 2 3 4 4 5 6 7 8
              8 7 6 5 4 3 2 2 1 2 2 3 4 5 6 7 8
                8 7 6 5 4 3 2 1 x 1 2 3 4 5 6 7 8
              8 7 6 5 4 3 2 2 1 2 2 3 4 5 6 7 8
            8 7 6 5 4 4 3 2 2 2 3 4 4 5 6 7 8
          8 7 6 6 5 4 4 3 3 3 4 4 5 6 6 7 8
        8 8 7 6 6 5 4 4 4 4 4 5 6 6 7 8 8
      8 8 7 6 6 5 5 5 5 5 6 6 7 8 8
    8 8 7 6 6 6 6 6 6 6 7 8 8
  8 8 7 7 7 7 7 7 7 8 8
8 8 8 8 8 8 8 8 8
```

**Figure 5**

Discrete octagon of diameter $2n + 1$ is the set of elements $\leq n$, $n = 1, 2, \cdots, 8$.

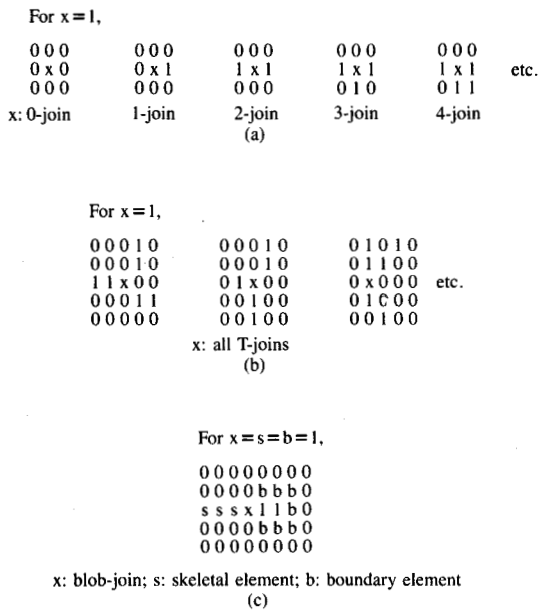



**Figure 6**

(a) *n*-joins; (b) T-joins; (c) a blob-join.

retains the "basic shape" of an entity. Unlike expansion or contraction, thinning both maintains the connectivity of an entity and preserves its holes (none are removed or added). This is an important distinction that we exploit in the inspection algorithms. It is useful (though imprecise) to think of thinning as a transformation that reduces elongated parts of entities to their centerlines and blobs that are approximately square or circular to their centers. The *4-*, *8-*, and *oct-thinning* used in the inspection algorithms are derived from 4-, 8-, and oct-contraction by imposing additional constraints on the removal of elements at each thinning step: (1) The global connectivity of entities is maintained and holes are preserved; (2) at each step, 4-thinning removes only elements with a zero 4-neighbor, whereas 8-thinning removes all elements with a zero 8-neighbor; (3) oct-thinning alternates 4- and 8-thinning. **Figure 3(b)** depicts the skeletons of the entities in Fig. 3(a) derived by 8-thinning.

*Element types and joins*

A *boundary element* is a nonzero element with a zero 8-neighbor; the *boundary* of an entity is the set of all its boundary elements. A *skeletal element* is a nonzero element that is necessary to maintain the connectivity of its 8-neighborhood; i.e., setting the element to zero breaks the connectivity between at least two other elements.

An *n-join* is a nonzero element with *n* nonzero 8-neighbors; a join can be of order 0 to 8. A *T-join* is a 3-join whose 8-neighborhood contains only skeletal elements. A *blob-join* is a skeletal element with an 8-neighbor that is not a skeletal element. Examples are given in **Figure 6.** The join order and type can be determined by using strictly local processing over 8-neighborhoods in an image; detection of join order and type plays a key role in the inspection algorithms.

*Generic method for analysis of printed circuit images*

Application of the above concepts to the analysis of printed circuit images led to the following observation: The local geometric and global topological correctness of typical circuit features can be inferred from the correctness of skeletal versions of the circuit features in a test image. This observation, in turn, led to the following generic method for the analysis of printed circuit images.

*Step 1* Transform and thin the test image in such a way that defects and good circuit features induce skeletal features that can be easily and reliably detected and classified.

*Step 2* Compile a detected feature list that records the position and type of all detected features.

*Step 3* Compare the detected feature list with a design feature list generated from circuit design data.

*Results* Features in the two lists that cannot be brought into correspondence imply defects.

The inspection algorithms described in this paper are instances of the generic method. Each algorithm uses a different thinning process designed so that a particular defect class induces a known corresponding class of skeletal features that can be easily and reliably detected. It turns out that the presence of 0-, 1-, T-, and blob-joins is sufficient to infer the existence of typical defects (as well as desired circuit features, such as pad-to-trace or trace-to-trace connections, trace ends, etc.).

The feature comparison method is quite flexible and powerful for two reasons. First, it is possible to define arbitrary correspondence criteria between arbitrary sets of detected and predicted features. In many cases, however, the following simple criterion probably will suffice: Detected and predicted features correspond if they are the same type and are within a given distance of one another. Features that cannot be brought into correspondence imply the existence of defects; the type of defect can be inferred from the type of feature. Second, it is possible to derive measures of global circuit correctness by combining the results of all isolated feature correspondences, e.g., spatial distortion, throughout the entire circuit layer.

For some circuit types and design rules it is not necessary to use reference data; the existence of certain skeletal features unambiguously implies defects. For example, if a circuit consists only of single width traces that must end in pads, 1-joins imply the presence of defects when the algorithm for verifying trace width described in Section 3 is used. However, for other classes of circuits, a comparison step is needed to distinguish between true defects and good circuit features that induce the same type of skeletal features.

• *Advantages over image comparison and design rule approaches*

Because our method does not compare a reference image and the test image pixel by pixel, it eliminates the need for the storage, generation, registration, and comparison of a reference image with the test image. Instead, a relatively small list of predicted feature types and locations is compared with a list of detected features in a straightforward way. Unlike direct image comparison, it is straightforward to incorporate context-dependent tolerances and attributes for features, e.g., pad location, type, and size. It is also easier to compensate for global distortion in the test image (e.g., scale and skew) because the inverse of the distortion function is applied to a relatively small set of feature variables, e.g., location, instead of the entire reference image. Finally, this method is relatively insensitive to local distortion and vagaries that can cause false alarms with direct image comparison. For example, irrelevant differences on the edges of traces or displacement of traces by a few pixels will not be flagged as errors (unless the minimum spacing rule is violated).

This method is a major improvement over design rule approaches because it can detect missing features and extraneous circuitization that looks like good features. In addition, unlike most design rule approaches, this method is not limited to verifying just minimum trace width and spacing; it can also verify pads, maximum trace width, and various trace connections, as well as detect isolated blobs, holes, etc. The new method is also capable of handling complex circuit features and circuit vagaries that can cause false alarms with design rule checkers. Finally, it can readily accommodate changes in circuit features and design rules that often require modification of inspection algorithms in design rule checkers.

## 3. Inspection algorithms

In general, the generic method can be applied to detecting and verifying the shape and size of a large class of feature types, including spacings, holes, lines, angles, corners, triangles, rectangles, octagons, and composites of these basic shapes. We make this concrete in this section by describing algorithms for

- Verifying minimum trace width and detecting open circuits.

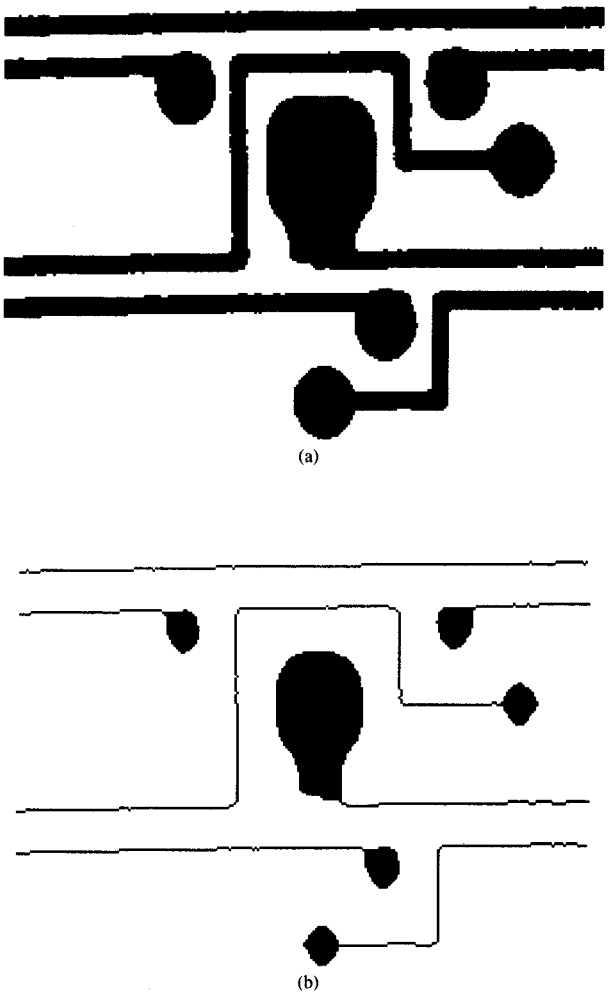



**Figure 7**

(a) Reference image and (b) its skeleton.

- Detecting excessive trace width.
- Verifying minimum spacing and detecting short circuits.
- Verifying pad position, area, shape, and trace-to-pad connections (including detection of spurious blobs).

However, the method is not limited to these algorithms or feature and defect types. We have developed numerous other algorithms for a variety of other feature and defect types.

We formally define each algorithm and describe in detail the associated transformation and thinning process. We do not discuss in detail generating the design feature list, compiling the defect feature list, or comparing the two lists because these operations are conceptually straightforward and easily implemented.

*Definitions*
$I$ = binary matrix, a discrete approximation to test image
$I'$ = binary matrix, output of contraction step
$I''$ = binary matrix, output of thinning step
$W$ = nominal maximum trace width, an odd integer
$w$ = minimum acceptable trace width, an odd integer $< W$

*Steps*
1: Oct-contract matrix $I$ $\lfloor w/2 \rfloor$ times (traces with width $< w$ are broken): result is contracted image $I'$
2: 8-thin image $I'$ $\lfloor W/2 \rfloor - \lfloor w/2 \rfloor$ times (traces of width $\leq W$ are skeletonized): result is binary matrix $I''$
3: Detect 1- and blob-joins in thinned image $I''$: result is list of detected feature types and locations
4: Compare list of 1- and blob-joins in $I''$ with design list

*Results*
1-joins in $I''$ not in design list $\Rightarrow$ trace width violations
1-and blob-joins in design list not in $I'' \Rightarrow$ missing features in $I$

**Figure 8**

Algorithm for verifying minimum trace width.

We also present the results of detecting simulated defects. For each simulation we added defects to the image shown in **Figure 7(a).** This reference image is a 200 by 256 binary image (matrix) of a 2.5 by 3.3-mm (0.1 by 0.128-in.) area of an inner plane layer of an IBM 3081 printed circuit board. It was acquired with a microscope, vidicon, and image digitizer; pixel size is approximately 0.013 mm square (0.5 mil square). The traces, small pads, and large pad are 7 pixels (0.089 mm), 25 pixels (32 mm), and 46 pixels (58 mm) wide, respectively. The skeleton of the reference image, obtained by 8-thinning sufficiently to skeletonize the traces, is shown in **Figure 7(b).**

• *Algorithm for verifying minimum trace width*

A summary of the four-step algorithm used to verify minimum trace width is illustrated in **Figure 8.** This algorithm can detect all instances of local trace width less than a programmable minimum; detect missing traces and a variety of spurious connections among traces, pads, and isolated blobs; detect holes in traces. The behavior of this algorithm is illustrated in **Figure 9** for the special case described there. In the contraction step, entities U, V, and W are contracted just enough to generate breaks in traces where the trace width is less than the minimum allowed. Note that entity W in Fig. 9(b) has broken up into the two distinct entities W′ and X′, whereas U and V contract to the two entities U′ and V′, respectively.

In the thinning step, the contracted traces are thinned just enough to produce their skeletons. This thinning does not cause the contracted entities to break up or disappear. It does, however, generate the two important feature types indicated in Fig. 9(d): (1) 1-joins wherever there exist minimum width violations; (2) blob-joins, the join points between trace skeletons and contracted pads. This does not imply, however, that there exists a one-to-one correspondence between 1-joins and width violations or between blob-joins and actual trace-to-pad connections. As one might expect, all features that look something like traces, i.e., elongated features whose width is less than the minimum allowable trace width, can generate 1-joins. For example, spurious blobs or large cracks in pads can generate 1- or blob-joins. The key point is that defective features that "look" like traces with minimum width violations will generate 1-joins; therefore, all such features can be detected.

The feature recognition step generates the detected feature list of all the 1-joins and blob-joins in the image. If the

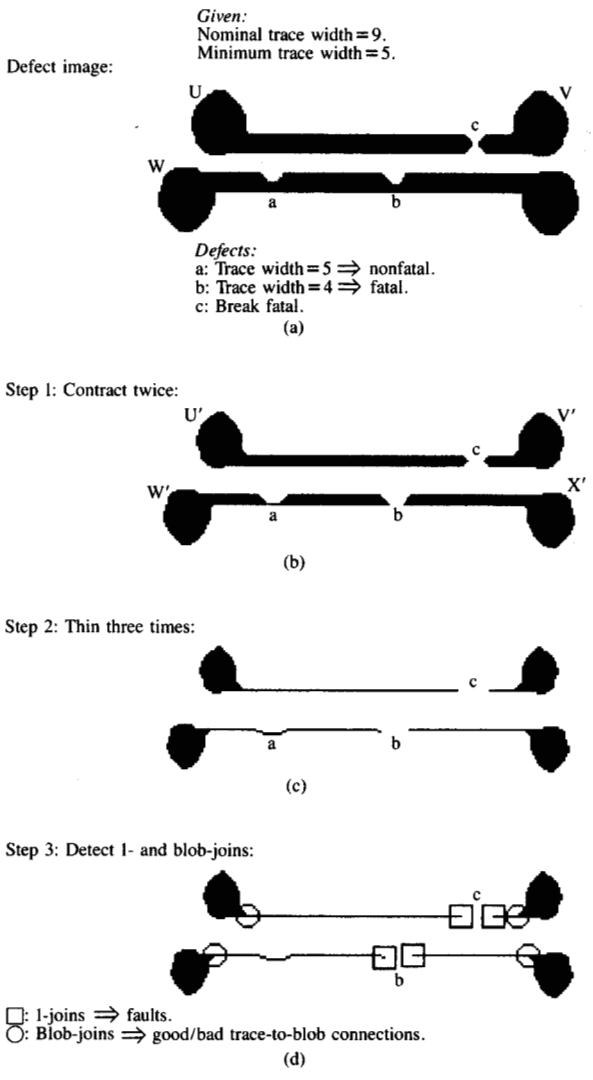



**Figure 9**

Illustration of algorithm for verifying minimum trace width: (a) defect image and specifications; (b) result of contracting defect image twice; (c) result of thinning contracted image three times; (d) detected 1- and blob-joins.

circuit type is such that 1-joins would not be generated for a good circuit, the comparison step is not needed: 1-joins imply the existence of minimum width violations. Here we assume that a trace is never completely missing. If a trace can be entirely missing, its absence can be inferred by using a comparison step because blob-joins will be missing in the detected feature list. Circuit types for which this is true are those consisting of single width traces, all of which must end in pads or features that are roughly circular (square) and whose diameter (width) is somewhat larger than the maximum allowable trace width. If 1-joins can occur, a comparison step is necessary.

In the comparison step the design feature list is compared to the detected feature list. Extraneous 1-joins in the detected feature list imply minimum trace width faults. Extraneous blob-joins in the detected feature list imply extraneous features (or excessive trace width). Missing 1- or blob-joins in the detected feature list imply missing features.

The result of applying the algorithm for verifying minimum trace width to an image containing simulated width reductions, open circuits, and holes is shown in **Figure 10.** The design rules used were as follows: Maximum trace width is 13 pixels (0.17 mm); minimum trace width is 5 pixels (0.063 mm); all traces must end in pads.

• *Algorithm for detecting excessive trace width*

A summary of the three-step algorithm used to detect excessive trace width is shown in **Figure 11.** Assuming that the maximum allowed trace width is $W$, this approach is capable of detecting local areas within a trace that contain a *discrete octagon* with diameters $\geq W + 2$ (see the Appendix for more details). For an illustration of this process, see **Figure 12.**

In the first step, entities are thinned just enough to skeletonize all traces with width $\leq W$. As shown in Fig. 12(b), octagon y and area a are completely thinned, whereas octagon z and area b are not. The presence of area b can be inferred from the presence of the extraneous blob-joins it induces in the thinned image. The positions of the extraneous and valid blob-joins are indicated in Fig. 12(c). In the comparison step, the detected feature list is compared with the design feature list. Blob-joins in the design feature list that are not in the detected feature list imply missing circuitization in the test image. Blob-joins in the detected feature list that are not in the design feature list imply excessive trace width in the test image.

The result of applying the algorithm for detecting excessive trace width to an image containing simulated trace bumps and fill-ins is shown in **Figure 13.** We assumed a nominal trace width of 8 pixels and a maximum trace width of 13 pixels.

• *Algorithm for verifying minimum spacing*

A summary of the four-step algorithm used to verify minimum spacing is shown in **Figure 14.** This algorithm can detect short circuits and spacings between distinct entities less than a programmable minimum. The behavior of this algorithm is illustrated in **Figure 15.** The first step in detecting spacing violations is to oct-expand all entities just

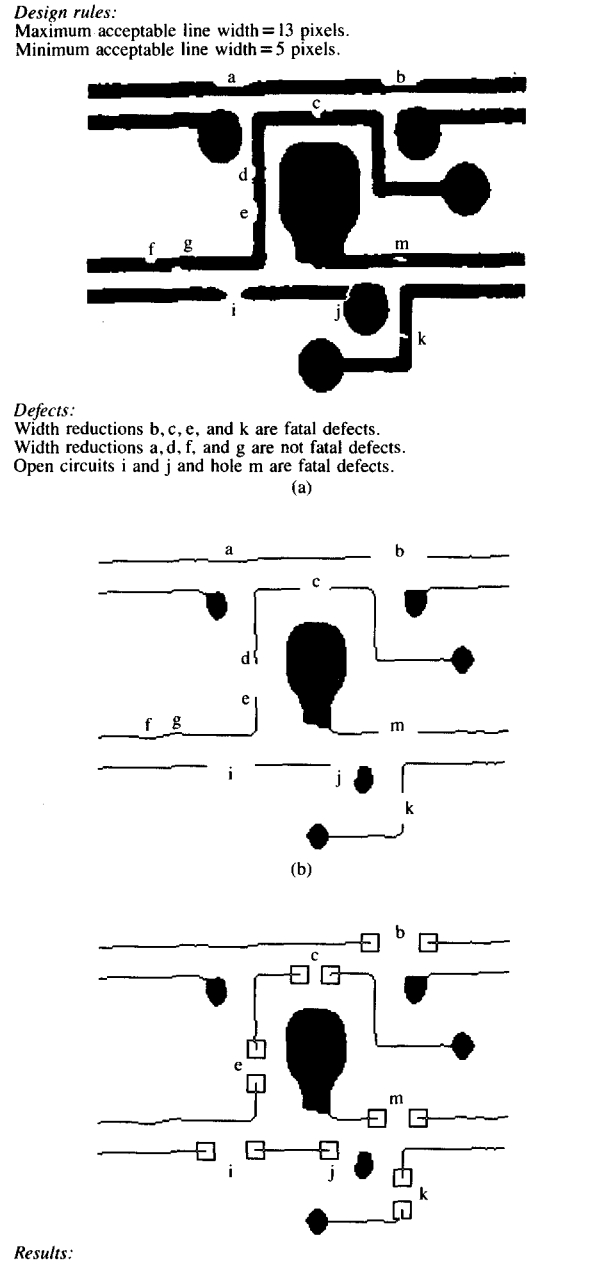



**Figure 10**

Detection of minimum trace width violations: (a) defect image with simulated defects; (b) defect image contracted and skeletonized; (c) detected defects.

*Definitions*
$I$ = binary matrix, a discrete approximation to test image
$I''$ = binary matrix, output of thinning step
$W$ = nominal maximum trace width, an odd integer

*Steps*
1: Oct-thin image $I$ $\lfloor W/2 \rfloor$ times (Traces of width $\leqq W$ are skeletonized): result is binary matrix $I''$
2: Detect blob-joins in $I''$: result is a list of detected blob-join locations
3: Compare detected blob-joins with design list

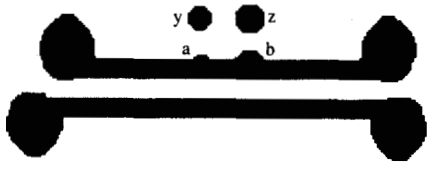
*Results*
Blob-joins in $I''$ not in design data $\Rightarrow$ excessive trace width

**Figure 11**

Algorithm for detecting excessive trace width.

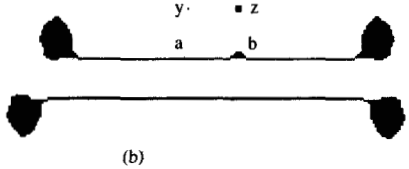
*Given:*
Trace area containing octagon with diameter > 11 is fatal.
Diameter of octagons y and z = 11 and 13, respectively (shown for comparison with defect areas a and b).

Defect image:



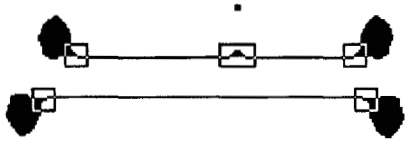

*Defects:*
Area a contains octagon with diameter = 11 $\Rightarrow$ nonfatal.
Area b contains octagon with diameter = 13 $\Rightarrow$ fatal.
(a)

Step 1: Thin five times:




(b)

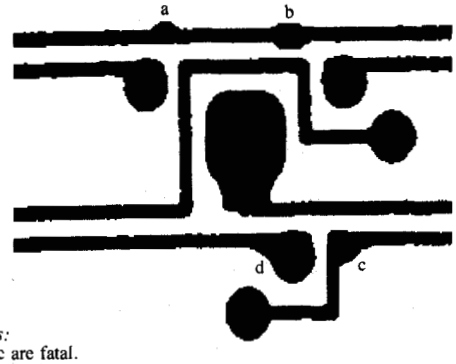
Step 2: Detect blob-joins:



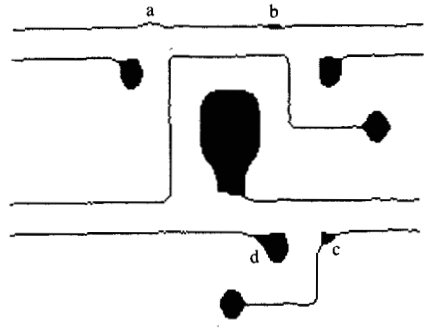
□: Blob-joins $\Rightarrow$ good/bad trace-to-blob connections.
(c)

**Figure 12**

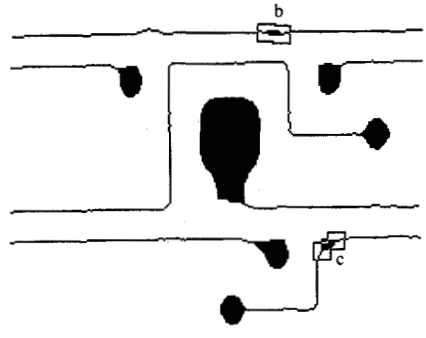
Illustration of algorithm for detecting excessive trace width: (a) defect image and specifications; (b) result of thinning defect image five times; (c) detected blob-joins.




*Defects:*
b and c are fatal.
a is not fatal.
Enlarged pad at d is not fatal.
(a)




(b)




*Results:*
□: Extraneous blob-joins at b and c imply defects.
(c)

**Figure 13**

Detection of areas of excessive trace width: (a) defect image with simulated defects; (b) skeletonized defect image; (c) detected defects.

enough so that entities touch (connect) at all locations where the spacing between them is less than the minimum allowed. The result of oct-expanding the defect image twice is shown in Fig. 15(b). Note that the bridge at c enlarges and a new bridge is formed at b; however, a bridge is not formed at a. The thinning step then skeletonizes the traces as well as all bridges with width less than or equal to fifteen. The key point to observe is that spacing violations like those in Fig. 15(a) generate extraneous T-joins (or blob-joins) in the expanded and thinned image. The defect features, along with

*Definitions*
$I$ = binary matrix, a discrete approximation to test image
$I'$ = binary matrix, output of expansion step
$I''$ = binary matrix, output of thinning step
$S$ = amount of thinning, an odd integer
$s$ = minimum acceptable spacing, an odd integer

*Steps*
1: Oct-expand matrix $I$ $\lfloor s/2 \rfloor$ times (spacings between entities $< s$ are filled): result is expanded image $I'$
2: 8-thin image $I'$ $\lfloor s/2 \rfloor + \lfloor S/2 \rfloor$ times (traces and bridges of width $S$ are skeletonized): result is binary matrix $I''$
3: Detect T- and blob-joins in $I''$: result is list of feature types and locations
4: Compare list of detected features with design list

*Results*
Features in $I''$ not in design list $\Rightarrow$ spacing violations in $I$
Features in design data not in $I'' \Rightarrow$ missing features in $I$

**Figure 14**

Algorithm for verifying minimum spacing.

good features, can be readily detected in the feature recognition step.

In the comparison step, the detected feature list of T-joins and blob-joins in the image is compared with a design feature list. Detected features not in the design list imply spacing violations or short circuits. Design features not in the detected features list imply missing circuitization.

The result of applying the algorithm for verifying minimum spacing on an image containing simulated short circuits and spacing reductions is shown in **Figure 16.** We assumed a design rule for minimum acceptable spacing of 5 pixels.

• *Algorithm for verifying pads*

Typically there are a variety of design rules for pads. For example, the design rules may constrain the size, shape, and position of pads, as well as limit the number and type of pad-to-trace connections. The thinning operation used in trace width verification can be used to partially verify pads. In particular, the existence and position of all blob-to-trace connections can be verified. Additional thinning can also be used to detect cracks, holes, and insufficient pad area. **Figure 17** illustrates how this can be accomplished: cracks, holes, and insufficient pad area induce extraneous 0-, 1-, T-, and blob-joins. Since extraneous joins do not match the design list, the defects that induce the joins are detected. An algorithm for verifying pads is given in **Figure 18.**

The result of applying the algorithm for verifying pads on an image containing artificial cracks, holes, and reduced area is shown in **Figure 19.** We assumed a minimum acceptable octagonal pad diameter of 15 pixels.

## 4. Hardware implementation of basic image processing operations

Contraction, expansion, thinning, and join detection are the basic image processing operations used in the inspection algorithms. These operations can be realized using sequences of *8-neighborhood operations* that map each pixel and its 8-neighbors in a binary matrix $I$ to a zero or a one. For algorithm development or for inspection tasks where execution time is not critical, these operations can be implemented on a general purpose computer or any of the commercially available image processing systems. However, both these approaches are far too slow to achieve full inspection of typical printed circuit layers in a few minutes or less (which requires a net throughput rate of several megapixels per second). For example, implementing the four

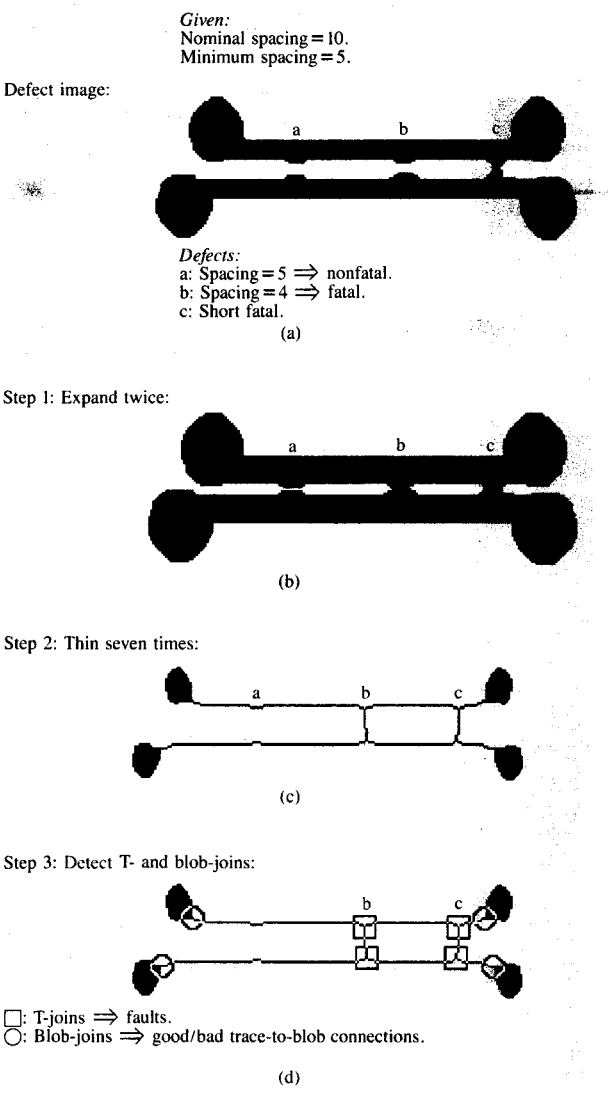



**Figure 15**

Illustration of algorithm for verifying minimum spacing: (a) defect image and specifications; (b) result of expanding defect image twice; (c) result of thinning expanded image seven times; (d) detected T- and blob-joins.

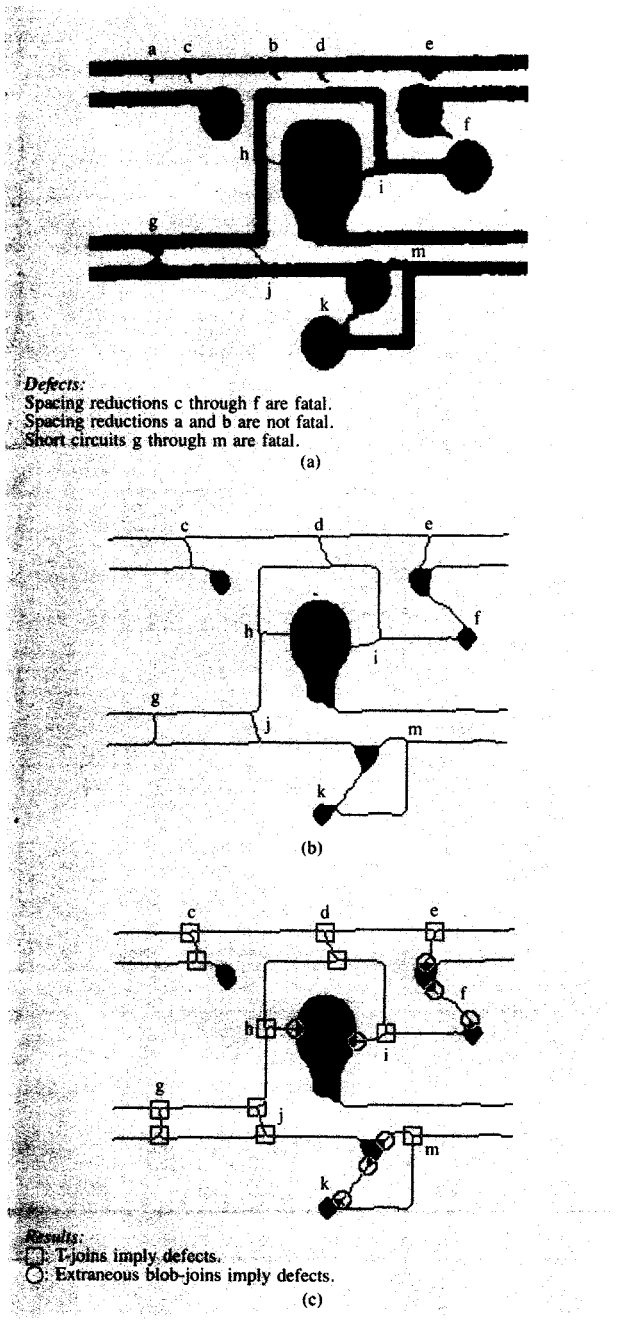



**Figure 16**

Detection of minimum spacing violations and short circuits: (a) defect image with simulated defects; (b) expanded and skeletonized defect image; (c) detected defects.

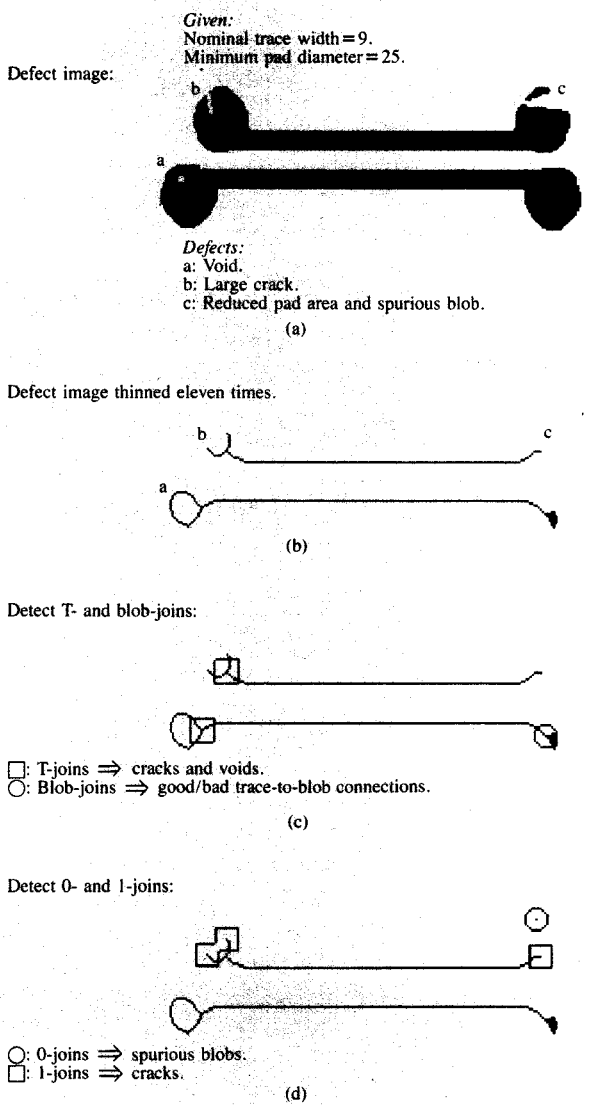



**Figure 17**

Detecting cracks, voids, and insufficient pad area: (a) defect image and specifications; (b) result of oct-thinning defect image eleven times; (c) detected T- and blob-joins; (d) detected 0- and 1-joins.

algorithms described in this paper requires about 150 8-neighborhood operations per image frame (for the design rules used in the simulations). Using a 0.025-mm (1-mil) square pixel, a 254 by 381-mm (10 by 15-in.) printed circuit layer contains about six hundred image frames. Assuming that an image processing system can achieve about five 8-neighborhood operations per second, it would take about three hundred minutes per layer to do all the required 8-neighborhood operations. Even assuming that an image processing system can achieve fifty 8-neighborhood operations per second, such an approach is still too slow by a factor of thirty. Experience has also taught us that implementation on a general purpose computer, in a high-level language like APL, is typically one to two orders of magnitude slower than that achievable with an image processing system.

An inspection system using a large number (hundreds) of frame-based image processors running in parallel could, in principle, achieve the necessary throughput rate of several megapixels per second. Fortunately, there is a better way. Sternberg [11] has shown how neighborhood operations can be implemented in hardware by *streaming* the elements of an image row by row through a simple *processing element* (PE) like the one shown in **Figure 20.** A sequence of $p$ neighborhood operations, e.g., $p$ contractions, can be implemented by *pipelining* (cascading) $p$ PEs, as also shown in Fig. 20.

Solid state imaging devices (especially linear arrays) are ideally suited as input image sources for the pipeline architecture. Using linear arrays, a circuit layer can be completely scanned by making multiple passes along one dimension, each pass offset from the previous one by slightly less than the width of the array. (Some overlap between passes is necessary to correctly process the image data on the borders.) By using this scanning strategy, image data can be continuously acquired and input, row by row, into the pipeline. Except for a small overhead in time to fill and empty the pipeline, the throughput of the pipeline is equal to the clock rate of the image source. It is well within the state of the art for linear arrays and this kind of pipeline architecture to operate at five to ten megapixels per second.

Using multiple, independently configured pipelines, all the inspection algorithms required for a particular application can be run in parallel. The throughput is limited only by the clock rate of the image source and pipeline. For example, the four algorithms described in this paper could be implemented with four independent pipelines containing a total of approximately 150 PEs. Assuming a conservative clock rate of five megapixels per second, it would take only thirty seconds to do the bulk of the image processing!

We have yet to discuss the generation of the detected feature list, a process we call *enumeration.* Each pipeline takes image data as input and produces multiple bit streams, each stream corresponding to the detection of a specific feature. In enumeration, all ones are converted to coordinate pairs corresponding to the location of the feature on the layer. A brute force approach to enumeration is to use an independent enumeration unit for each bit stream. Conceptually, the enumeration unit is simply one more stage at the end of the pipeline.

The list comparison can be performed on a general purpose computer using standard sorting techniques. It is important to note that because the scan pattern is known ahead of time, the feature data are almost entirely presorted. The data are not entirely presorted because the inherent uncertainty in the position of features randomizes the order of features that are within a few rows or columns of one another. It is also important to note that most of the list comparing can be done in parallel with the generation of the feature list. If list comparison is overlapped with the physical

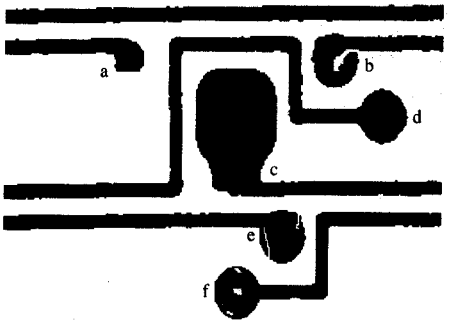
*Definitions*
$I''$ = binary matrix, result from thinning step in algorithm for minimum trace width verification
$P$ = binary matrix, result of additional thinning
$W$ = nominal maximum trace width, an odd integer
$d$ = diameter of minimum allowed circular pad area, an odd integer, $d \geq W + 2$
$r = (d - 3)/2$, $d > 3$

*Step*
1: Detect 0- and T-joins in $I''$: result is list of detected features in $I''$
2: Oct-thin $I''$ by amount $r - (W - 1)/2$: result is matrix $P$
3: Detect 0-, 1-, T-, and blob-joins in $P$: result is list of detected features in $I''$ and in $P$
4: Compare detected feature list with design list

*Results*
0-joins not in design list $\Rightarrow$ extraneous metallization; broken features
1-joins not in design list $\Rightarrow$ cracks; insufficient pad area
T- and blob-joins not in design list $\Rightarrow$ cracks; voids; possibly hard shorts

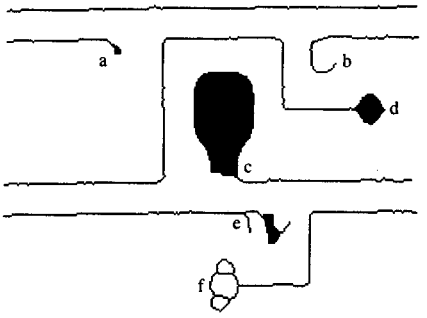
**Figure 18**

Algorithm for verifying pads.




*Defects:*
Insufficient pad area at b and f is fatal.
Cracks at e and reduced area at a are not fatal.

(a)




*Results:*
1-join and missing blob-join at b imply fatal defect.
T-joins and missing blob-join at f imply fatal defect.
1-joins and extraneous blob-join at e imply defect.

(b)

**Figure 19**

Detection of pad defects: (a) defect image containing simulated defects; (b) skeletonized defect image and results.

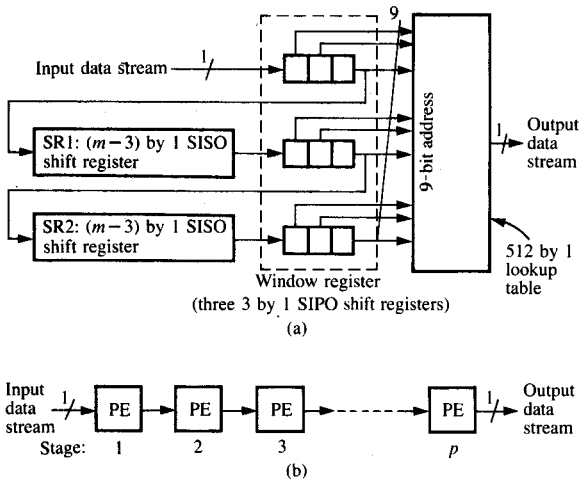



**Figure 20**

Hardware implementation of basic image procession operations: (a) basic processing element (PE); (b) pipeline of $p$ processing elements.

handling of a layer and the image processing, we estimate that list comparison does not add significantly to the overall inspection time.

## 5. Concluding remarks

This paper has described a new method for the analysis of printed circuit images. As the algorithms presented here demonstrate, the new method is a reliable and flexible way of detecting a variety of common defect types in discrete, binary images of printed circuit features. However, the method is by no means limited to these particular algorithms or defect types. The method can be applied to detecting and verifying the shape and size of a wide class of feature types, including lines, angles, corners, triangles, rectangles, octagons, and composites of these basic shapes. However, there does not exist a "magic formula" for generating an inspection algorithm as a function of feature type; this still requires ingenuity. For example, we have developed algorithms for verifying the existence and diameter of holes, checking clearance between holes and other features, handling traces and pads of different sizes, verifying that a local feature is connected to other arbitrarily distant features, etc.

In addition to its flexibility, this method is well suited for high-speed implementation using pipelines of simple processing elements. The speed of processing is roughly proportional to the number of independent pipelines used and the number of PEs per pipe. Independent pipelines can be configured to process different image areas in parallel, to implement different algorithms in parallel on the same image area, or any combination of the two.

Finally, we note that a reliable, flexible, efficient, and cost-effective system is more than just a set of inspection algorithms—no matter how elegant the algorithms may be. On the research side, each inspection problem has to be thoroughly analyzed and experiments conducted to determine the limits and capabilities of this method. On the engineering side, reliability, throughput, and cost considerations will dictate the specifics of a given implementation.

## Appendix

The image processing operations used in the inspection algorithms can be realized by boolean operations between images and *8-neighborhood operations*. An 8-neighborhood operation is a binary image to image transformation that maps in parallel all 8-neighborhoods of the input image to a one or a zero. It is often useful to define the mapping in terms of a set of *templates*. A single template is a three by three array with a specific configuration of zeros, ones, and don't cares. An 8-neighborhood operation is realized by comparing the 8-neighborhoods of an image with a template set: the result is one if the 8-neighborhood matches at least one of the templates in the set; otherwise the result is zero.

- *Discrete distance*

In an analog image, we measure trace width, spacing, pad diameter, etc., using the usual euclidean metric. However, in a discrete, binary image, we must use a discrete approximation to the euclidean metric. In the inspection algorithms we infer distance using oct-expansion and contraction. The combination of expansion and contraction used implicitly defines a discrete distance measure. The shape of an entity generated from an isolated element by $n$ expansion steps can be thought of as a *discrete distance map* of order $n$. For example, if a discrete octagon of arbitrarily large diameter is centered at an element $I(i, j)$, then the distance between $I(i, j)$ and any other element $I(k, l)$ is equal to the value of the element in the discrete octagon overlaying the element $I(k, l)$. Other types of distance maps we could employ include diamonds, rectangles, squares, ovals, and sixteen-sided figures. The inspection algorithms can be modified to employ any combination of expansion or contraction best suited to a particular application.

- *Expansion (contraction)*

The operations of 4- and 8-expansion (contraction) can be realized using a single step of template matching. 4- and 8-expansion are realized by setting an element to one if it has a 4- or 8-neighbor equal to one, respectively. 4- and 8-contraction are realized by setting an element to zero if it has a 4- or 8-neighbor equal to zero, respectively. Oct-expansion or contraction is realized by alternating 4- and 8-expansion or contraction, respectively.

- *Thinning*

We use a thinning that preserves the *homotopy tree* of an image. This means, among other things, that the existence

and connectivity of all entities and of all the holes is preserved [12]; 4-, 8-, and oct-thinning can be realized using 8-neighborhood operations with the template set shown in **Figure 21.** However, unlike contraction, entities cannot be thinned uniformly from all directions with a single 8-neighborhood operation because entities with a width of two elements would be completely removed. This problem can be overcome using the following four-step algorithm for 8-thinning.

*8-Thinning algorithm based on template matching*

*Step 1* Remove all points in image $I$ with 8-neighborhoods that match top template set.
Result: image $B$ ($I$ thinned from the top).
*Step 2* Remove all points in image $B$ with 8-neighborhoods that match right template set.
Result: image $C$ ($I$ thinned from top and right).
*Step 3* Remove all points in image $C$ with 8-neighborhoods that match bottom template set.
Result: image $D$ ($I$ thinned from top, right, and bottom).
*Step 4* Remove all points in image $D$ with 8-neighborhoods that match left template set.
Result: image $I$ 8-thinned.

The 4-thinning algorithm is derived from 8-thinning by imposing an additional constraint that suppresses the removal of elements with a zero 8-neighbor but not a zero 4-neighbor. This constraint is realized by first deriving an image $I'$ from the input image $I$ by setting an element of $I'$ to one if and only if the corresponding element in $I$ has a zero 8-neighbor but not a zero 4-neighbor. Then image $I'$ is ORed element by element with the result of each step in the 8-thinning algorithm prior to the next step.

If only the basic templates are used, the thinned result will often contain spurious skeletal segments induced by relatively small features protruding from the boundary of an entity (e.g., artifacts of the sampling process). The *trimming* templates are used primarily to "clean" the noise from the skeleton by suppressing the growth of spurious skeletal segments. The trimming templates can also be used by themselves to *trim* all 1-joins from entities without thinning the entities.

• *Feature detection*

Detection of $n$-joins and T-joins is achieved by using one set of templates applied once in parallel over an image. In effect, a set of templates used to detect $n$-joins counts the number of ones in an 8-neighborhood: the result is one if the count is $n$; zero otherwise. A template for detecting T-joins can be constructed by appropriately combining a template set for 3-joins with the template set illustrated in Fig. 21 for extracting skeletal points with a zero 4-neighbor. Blob-joins cannot be detected using one set of templates applied once in parallel over an image. However, they can be detected using the following algorithm.

For a = 1; x arb.:

| Top | Right | Bottom | Left |
|---|---|---|---|
| x00 / 1a0 / x1x | x1x / 1a0 / x00 | x1x / 0a1 / 00x | 00x / 0a1 / x1x |
| 00x / 0a1 / x1x | x00 / 1a0 / x1x | x1x / 1a0 / x00 | x1x / 0a1 / 00x |
| x0x / 1a1 / x1x | x1x / 1a0 / x1x | x1x / 1a1 / x0x | x1x / 0a1 / x1x |

Basic templates

| | | | |
|---|---|---|---|
| 000 / 0a0 / 100 | 100 / 0a0 / 000 | 001 / 0a0 / 000 | 000 / 0a0 / 001 |
| 000 / 0a0 / x1x | x00 / 1a0 / x00 | x1x / 0a0 / 000 | 00x / 0a1 / 00x |

Trimming templates

For a = 1; at least one x and y = 1:

| | | | |
|---|---|---|---|
| xxx / 0a0 / yyy | y0x / yax / y0x | yyy / 0a0 / xxx | x0y / xay / x0y |

For a = 1: x arb.:

| | | | |
|---|---|---|---|
| xxx / 0ax / 10x | 10x / 0ax / xxx | x01 / xa0 / xxx | xxx / xa0 / x01 |
| 000 / 010 / 0x0 | 000 / x10 / 000 | 0x0 / 010 / 000 | 000 / 01x / 000 |

Templates for extracting skeletal points

**Figure 21**

Templates for thinning, trimming, and extracting skeletal points.

*Algorithm for detecting blob-joins*

*Step 1* Extract skeletal points with a zero 4-neighbor: result is $S$.
*Step 2* 8-thin $I$: result is $A$.
*Step 3* Remove $A$ from $I$: result is $B = I \wedge \sim A$.
*Step 4* 8-expand $B$: result is $C$.
*Step 5* Blob-joins in $I = S \wedge C$.

($\wedge$ and $\sim$ are the boolean operations "and" and "not," respectively.)

• *Detecting discrete octagons*

The algorithms for detecting excessive trace width and verifying that all traces end in discrete octagons of a minimum size are based on the following property of oct-contraction:

*Given that the original entity is oct-contracted* n *times, all remaining elements of the contracted entity must be the center of a discrete octagon in the original entity with a diameter* $\geq 2n + 1$.

As a consequence, oct-thinning tends to have the following property:

*Given that the original entity is oct-thinned* n *times, a nonskeletal point with a skeletal 8-neighbor in the thinned entity is the center of a discrete octagon in the original entity with a diameter* $\geq 2n + 1$.

There do exist cases for which this is not true; however, these cases seem to be characterized by join types that can be easily detected. The algorithms for detecting excessive trace width and verifying pad area exploit this property of oct-thinning. Note, however, that these algorithms detect discrete octagons with a diameter $\geq 2n + 3$—not $2n + 1$—by thinning $n$ times and detecting blob-joins. This is because a discrete octagon of diameter $2n + 1$ (and some areas between octagons of diameter $2n + 1$ and $2n + 3$) will be completely skeletonized with $n$ oct-thins; hence these areas cannot be detected by oct-thinning $n$ times and detecting blob-joins. However, it is true that all areas that contain octagons with a diameter $\geq 2n + 3$ can be detected this way.

## References


1. Donald P. Seraphim, "A New Set of Printed-Circuit Technologies for the IBM 3081 Processor Unit," *IBM J. Res. Develop.* **26,** 37–44 (January 1982).
2. A. J. Blodgett and D. R. Barbour, "Thermal Conduction Module: A High-Performance Multilayer Ceramic Package," *IBM J. Res. Develop.* **26,** 30–36 (January 1982).
3. R. Chin, "Automated Visual Inspection: A Survey," *IEEE Trans. Pattern Analysis & Machine Intelligence* **PAMI-4,** 559–562 (November 1982).
4. R. Kruger and W. Thompson, "A Technical and Economic Assessment of Computer Vision for Industrial Inspection and Robotic Assembly," *Proc. IEEE* **69,** 1526–1529 (December 1981).
5. T. Pavlidis, *Structural Pattern Recognition*, Springer-Verlag New York, Inc., 1977, Chs. 3, 9.
6. A. Rosenfeld and J. Pfaltz, "Distance Functions on Digital Pictures," *Pattern Recognition* **1,** 33–61 (1968).
7. A. Rosenfeld, *Picture Languages*, Academic Press, Inc., New York, 1979.
8. J. Serra, *Image Analysis and Mathematical Morphology*, Academic Press, Inc., New York, 1982, Chs. 1–8.
9. E. Abbott, M. Hegyi, R. Kelley, D. McCubbrey, and C. Morningstar, "Computer Algorithms for Visually Inspecting Thick Film Circuits," *Proceedings of RI/SME Conference on Applied Machine Vision*, Memphis, TN, February 1983.
10. M. Ejiri, T. Uno, M. Mese, and S. Ikeda, "A Process for Detecting Defects in Complicated Patterns," *Computer Graph. & Image Process.* **2,** 326–339 (1973).
11. S. Sternberg, "Biomedical Image Processing," *IEEE Computer* **16,** 22–28 (January 1983).
12. J. Serra, op. cit., pp. 187–206.


*Received June 20, 1984; revised August 27, 1984*

**Jon R. Mandeville** *IBM Research Division, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Mandeville is a research staff member in the industrial machine vision project of the Manufacturing Research Center. He joined IBM in 1982 after receiving the Ph.D. in electrical engineering from Stanford University, Palo Alto, California. His academic study at Stanford focused on optical and digital signal and imaging processing. He received a B.S. in electrical engineering from the University of Washington in 1974 and an M.S. in electrical engineering from Stanford University in 1978. While still a graduate student at Stanford, he worked as an academic associate for the IBM San Jose Research Laboratory in the experimental systems group. His work there focused on digital filtering techniques for enhancing the quality of black and white documents reproduced from scan data generated from devices such as flying spot scanners and linear solid state imaging arrays. This work formed the basis for his Ph.D. thesis. His current work is the application of optical and digital techniques for automating visual inspection tasks in the manufacturing environment.