Per-Erik Danielsson

An Improved Segmentation and Coding Algorithm for Binary and Nonbinary Images

This paper presents a new segmentation and coding algorithm for nonbinary images. The algorithm performs contour coding of regions of equally valued and connected pixels. It consists of two distinct phases: raster scanning and border following. In this sense it is similar to algorithms presented by Kruse. However, the algorithm of this paper is considerably improved since it correctly segments truly nonbinary images. The basic idea of the algorithm is to "coat" (color, label) the borders (the cracks) between the regions from both sides in two separate border-following procedures called island following and object following. Thus, all adjacencies between the objects are systematically explored and noted. Furthermore, the raster scanner, which exhaustively searches the image for new regions, can easily determine from existing/nonexisting coating which boundaries have been traced out and which have not. The algorithm can be considerably simplified for the binary image case.

1. Introduction

In this paper, segmentation means separation of objects or regions of a digitized image. An object is a set of connected pixels having the same value, and we assume that the pixels are embedded in the usual Cartesian grid. The algorithm to be presented has the image pixels as input data and delivers a coded version of the image as output, where each object is defined by its chain-coded contour. In addition to this, the output data also contain the topological graph of the original image where the objects are the nodes of the graph and adjacencies between two objects are the branches.

A segmentation procedure like this is a valuable ingredient in many image processing and pattern recognition problems. It follows naturally after such preprocessing steps as noise suppression, filtering, and thresholding or after more sophisticated relaxation procedures where pixels, or rather clusters or regions of pixels, have been assigned a label from the set $\{0, 1\}$ (the binary case) or from a larger set $\{0, 1, 2, \cdots, n\}$ (the nonbinary case).

Several authors have been dealing with the segmentation problem. Morrin [1] segments a binary image by a combination of raster scanning and boundary following. As soon as the raster scan encounters a boundary, the algorithm due to Morrin starts to follow the boundary, peeling off one layer of pixels after another until the object is exhausted. Raster scanning is then resumed. Only the first boundary trace is stored as a contour.

The advantage of Morrin's method is that it only requires a binary memory for the image. Its disadvantage is that the rather time-consuming boundary-following operation has to be carried out not only for the boundary of the objects but for the interior pixel layers as well.

Cederberg [2, 3] and Danielsson [4] avoid the random-addressing, border-following mode completely. Instead, the contours are kept track of as they appear, grow, and disappear from one line to the next. This requires a substantial amount of bookkeeping. The great advantage of these methods is that no image memory at all is required. As a consequence, the coding of the image can be done directly during the raster scan input.

In two recent papers Kruse [5, 6] has presented segmentation algorithms for both binary and nonbinary images. Like Morrin [1] Kruse uses both raster scanning and border following. But to avoid the time-consuming layer by layer peel-off, a multistate memory is utilized. See Fig. 1(a). The original pixels of the binary image are labeled from the set

{0, 1} while the segmented image is labeled from the set $\{A, B, C, \cdots\}$. As soon as a new boundary is encountered, raster scanning is interrupted. An unused label is drawn from the set $\{A, B, C, \cdots\}$, and during a border-following procedure this label is written into the boundary pixels inside the newly encountered object, as shown by Fig. 1(b).

Also, during border following, pixels immediately inside and simultaneously west of the border are tagged, which in Fig. 1(b) is symbolized by a circle around the pixel values. The tag occupies one extra bit-plane in the multistate memory. In Fig. 1(b) the background object is labeled A; object B has been border-followed; the raster scan has been resumed and has been continued to the fourth line from the top. By using the knowledge that $B = 1$, when mapped to the original of Fig. 1(a) the raster scanner can label all pixels inside the B-object for each run. Furthermore, the tag enables the raster scanner to distinguish between a case in which it is leaving the B-object and entering the A-object from a case in which it is leaving the B-object and entering a new object (a hole inside B). In the former case the A-label is fetched from a stack. In the latter case a new border-following procedure takes place.

The stack is an important ingredient in Kruse's algorithms. It is pushed during raster scanning when a new label from the set $\{A, B, C, \cdots\}$ is encountered and popped when a tagged pixel is hit. Obviously, the stack contains information about the nesting levels of the different objects. For the simple case of Fig. 1(a) the nesting tree is depicted in Fig. 1(d).

The label map that transforms Fig. 1(b) into Fig. 1(a) is shown by Fig. 1(c).

In [6] Kruse extended the above method to the nonbinary case. It is shown that the algorithm works quite well on nonbinary images of the type shown by Fig. 2(a), which is transformed to the appearance of Fig. 2(b) and delivers the label map of Fig. 2(c) and the nesting tree of Fig. 2(d).

Unfortunately, the algorithm [4] does not work very well on the more general kind of nonbinary images where an "island" contains several separate regions, as shown by Fig. 3. The "lake" of 0's inside the larger island will pass undetected as a separate object when in the fifth line from the top the raster scanner pops the stack, giving top-of-stack = A. Now, since according to the map $A = 0$ and since the next pixel has the original label 0, the raster scanner assumes it has returned to the surrounding A-region.

Another deficiency is that the outer contour of the island is not retrieved. Thus, full geometric information about the A-object is not available in the encoded data. Finally, as can

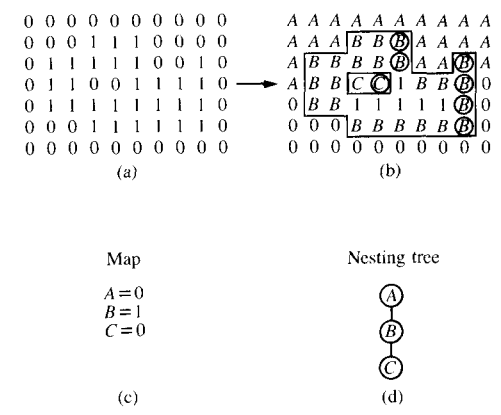

Figure 1 Kruse's algorithm operating on a binary image.

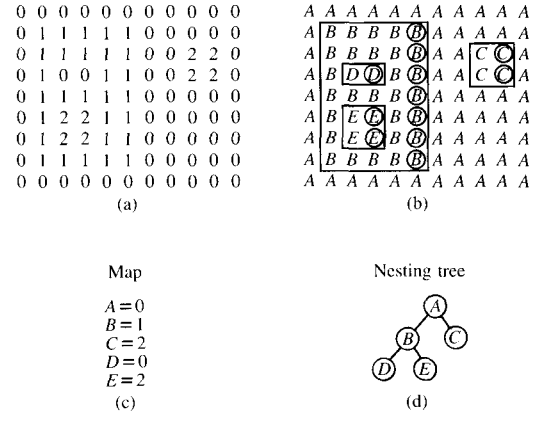

Figure 2 Kruse's algorithm operating on a pseudo-binary image.

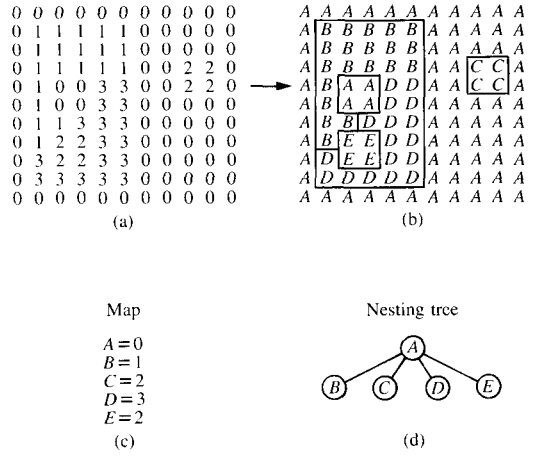

Figure 3 Kruse's algorithm operating on a truly nonbinary image. Labeling is incorrect and topology detection incomplete.

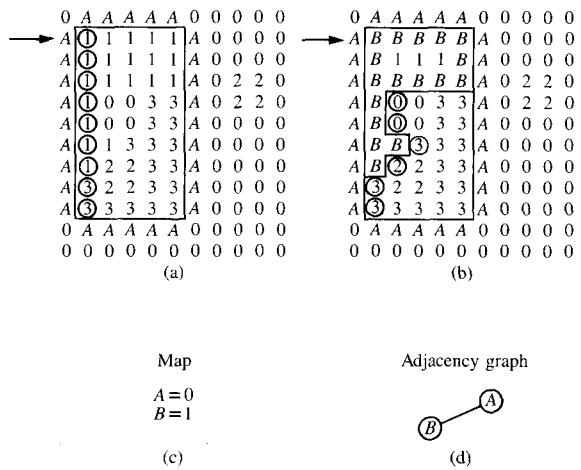

Figure 4 The two first steps of the new algorithm.

be seen from the nesting tree of Fig. 3(d), all four objects B, C, D, and E will be registered as being inside A without revealing the full topological adjacency conditions of the original image.

It should be noted, however, that the algorithm does deliver a contour code which is sufficient for reconstructing the original image.

2. The improved algorithm

We start describing the improved algorithm for the more complicated nonbinary case. For the binary case a simplified version can be used, as explained in Section 6. As in Kruse's algorithm, raster scanning is the background mode of the improved algorithm. As soon as the raster scanner hits a pixel with a value from the original set {0, 1, 2, · · ·} that is different from its current map entry, the algorithm switches to border following. (Precise rules for this algorithm are in the flowcharts of Figs. 6–8.)

• *Island following*

For the image of Fig. 3(a) the first hit happens in the second line where the raster scanner is currently reading 0's knowing that $A = 0$ according to the existing map entry. The appearance of a 1 in the second pixel of this line interrupts raster scanning. We have obviously encountered an island inside the A-object, and this island is now traced out by following its outermost border. During this operation we collect the chain code of the border, preferably as the chain of two-bit links that exactly follows the crack between the background object A and the island.

During border following we insert new labels A in all pixels immediately outside the crack [Fig. 4(a)]. The A-labels to the east of the crack are to be used subsequently by the raster scanner to avoid following this crack again as a new island. The A-labels to the south, west, and north of the crack will enable the subsequent border-following procedures to establish adjacency conditions. Also, during the border following we tag pixels which are immediately inside and simultaneously east of the crack as an indication that the border has been followed on its western side. This will help us to avoid unnecessary island following when the raster scanner encounters the crack in subsequent lines. To the west of a tagged pixel there is consequently always a label from the new set, and one may suspect that this label alone could serve as an indicator for "island following done." However, for one-pixel-wide objects there is a label from the new set to the west of the crack before any border following has been done as a result of the neighboring crack.

• *Object following*

So far the original image of Fig. 3(a) has been converted to the shape of Fig. 4(a). We have completed the island following and are back at the point where we first found the new island. We now start the object following of the object labeled 1 in the original image. This time we trace the inside of the encountered crack. We have drawn a new label B which is now to be written into all pixels next to the crack and inside the object. Incidentally, with the small objects we have in our running example this writing almost fills the whole interior. For larger objects there will be large areas with the original labels left.

During object following we delete internal tags but tag pixels outside and immediately east of the crack in those instances where the pixel east of the crack still has an original label, *i.e.*, a number instead of a letter. This will help us subsequently to avoid island following when the raster scanner encounters the objects originally labeled 0 and 2.

We have now reached the state shown by Fig. 4(b). Note that the tags of Fig. 4 serve a completely different purpose from those in Kruse's algorithm. Also, we have started to build up our label map and the adjacency graph during this first border following, as shown by Figs. 4(c) and (d), respectively.

• *Further iterations*

Raster scanning is now resumed, and using the available map information it goes on uninterrupted until it hits the first pixel labeled 2 in the fourth line. Since this pixel, which is to the east of the crack, is untagged, first island following and then object following take place.

Next the raster scanner halts in the fifth line when the first pixel labeled 0 is hit. Because of the tag of this label, the algorithm immediately switches to object following with a

new map entry, $D = 0$, *i.e.*, bypassing island following. After a short raster scan trip, a new object-following procedure takes place with $E = 3$. After this step the image has the appearance of Fig. 5(a).

In Figs. 4 and 5 we have deliberately avoided filling in the objects with new labels during raster scanning. This saves some writing time but is otherwise of no significance to the algorithm.

During all object-following procedures, adjacencies between the followed object and objects on the other side of the crack are noted in the adjacency graph, which is the necessary generalization of the nesting tree of Fig. 3. Naturally the border follower should economize on this notation work and only augment the graph when the sequence of labels outside the crack changes. For instance, when the object $E = 3$ is followed, the first adjacency is to the object D. With clockwise movement the next adjacency is B. Remembering this last adjacency condition we do not have to note B again when we find the second pixel $= B$ outside the crack. Next we note that the object A is adjacent to our object E, after which we follow the crack a long time without any notations at all in the graph since the pixels outside the crack are A's consistently. When we encounter B again, this is a change outside the crack so we note that E is adjacent to B without knowing that this has been noted already. Then, the labels 2 are neglected since only labels from the set $\{A, B, C, \cdots\}$ can exist in the graph. Another instance of B outside the crack is then found and noted for the third time, and finally a second instance of D-adjacency is noted.

If the graph is updated with a procedure like this, it should, of course, be immediately followed by a cleaning routine that reduces the branches of the object E to a proper set.

After the object-following results in Fig. 5(a), raster scanning is resumed. It is halted for the last time at the eighth line where the object $F = 2$ is traced out. The final result is shown by Fig. 5(b), with the complete map and graph in Figs. 5(c) and (d), respectively.

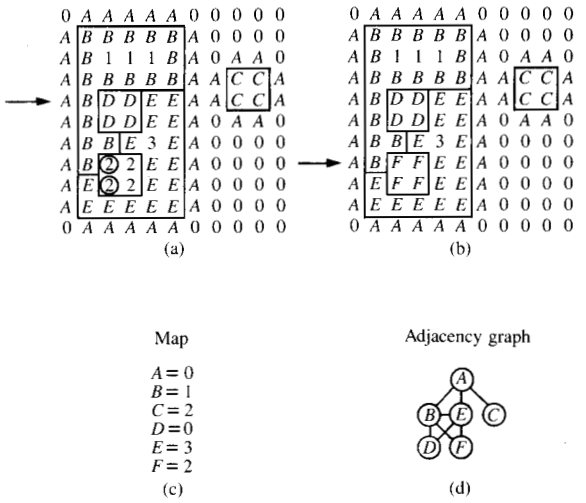

Figure 5 The final result of the new algorithm.

3. Flowchart description

A rather complete description of the algorithm is given by the flowcharts in Figs. 6, 7, and 8. The reader should have no difficulties in interpreting the different steps therein and identifying them with the verbal description given in the preceding section. However, a few comments may be appropriate.

Around the border of the full image we adopt the convention that there is a frame with the imaginary label "frame," which is included in the new set NS $= \{A, B, C, \cdots,$

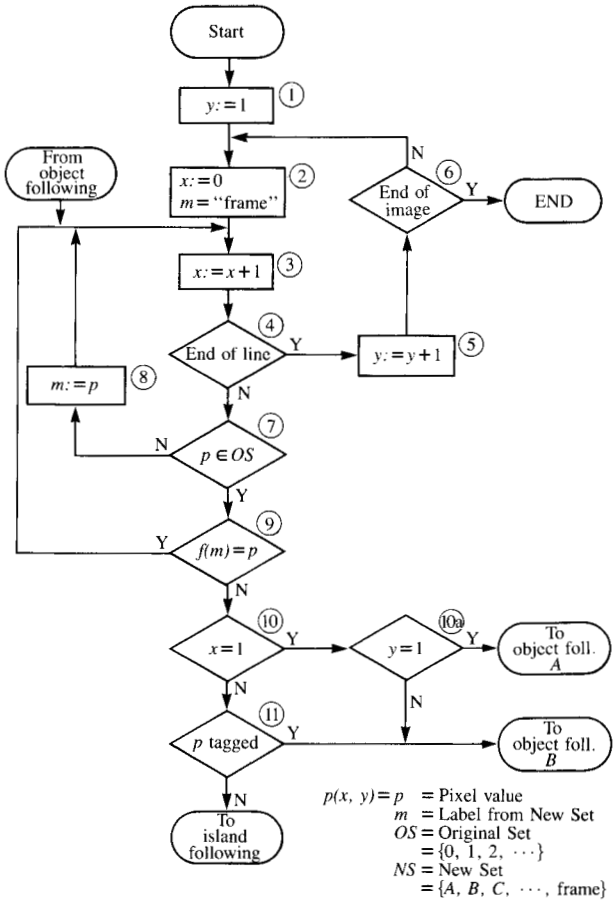

Figure 6 The raster-scanning part of the algorithm.

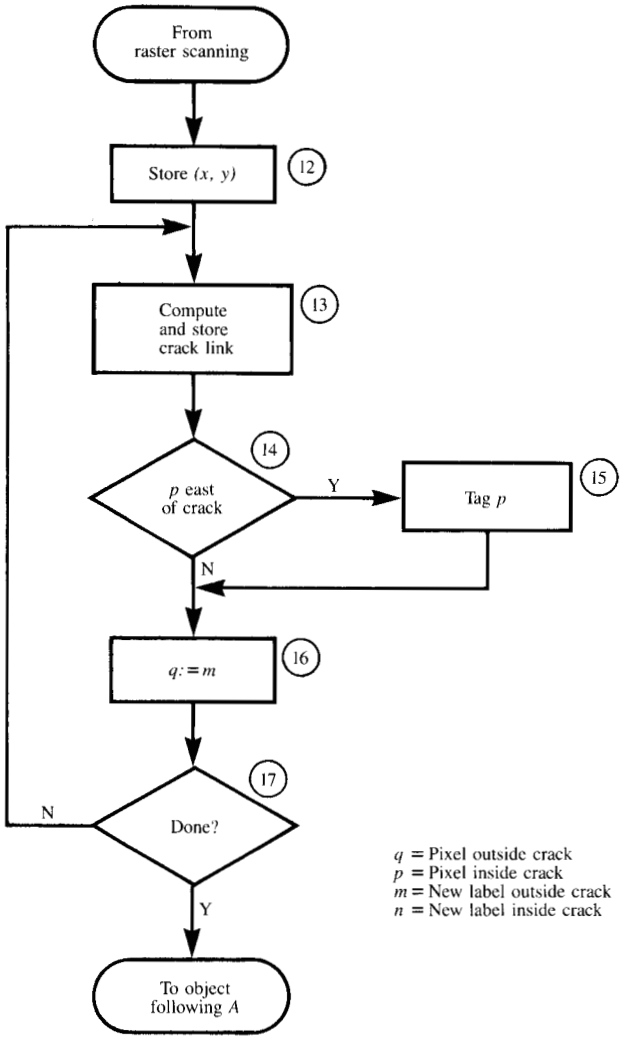

Figure 7 The island-following part of the algorithm.

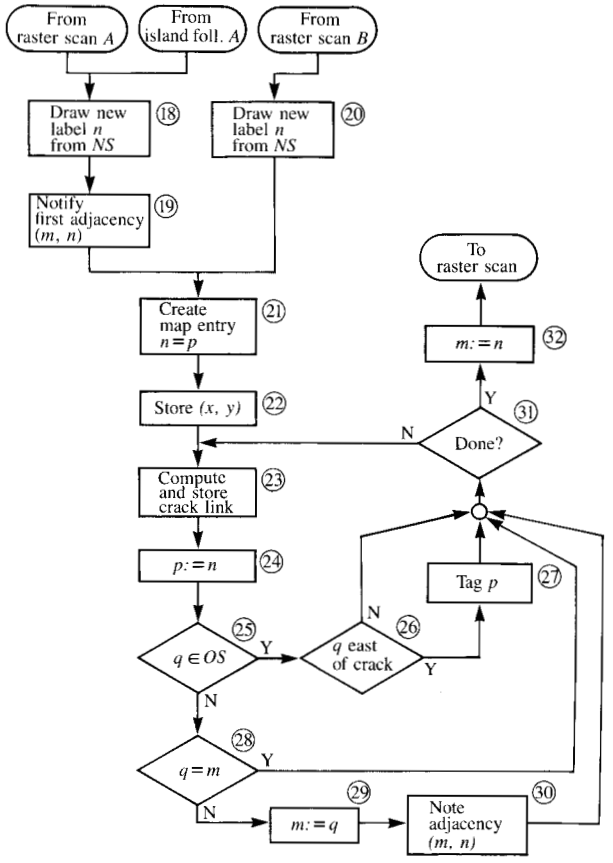

Figure 8 The object-following part of the algorithm.

"frame"}. The variable m which is continuously updated to the present object label during raster scanning, box 8, is consequently set to "frame" at the beginning of each line in box 2. Also, "frame" is always the root (not shown) of our adjacency graph in the previous Fig. 5(d). The frame concept was not used in the previous section since at that time we did not bother at all with picture boundary problems.

The first pixel encountered by the raster scanner at $x = 1$, $y = 1$ will always trigger object following. If this object A is the only one that touches the frame (implying that all other objects are directly or indirectly inside A), the whole image boundary will be labeled with one layer of A's during this first object following.

Box 13 in Fig. 7 computes and stores the crack links for the island outline. If the island consists of only one object, the same crack is shortly thereafter traced out in box 23 of Fig. 8. The first contour is stored as one of possibly several islands in the surrounding object, while the second identical contour is stored as the outline of a new object within the new island.

All adjacency conditions are registered by box 30. However, to be able to relate one island in a surrounding object to the proper set of objects constituting this island, there is also a "first adjacency notation" in box 19. Here, a link is created from the surrounding object to one of the objects in the corresponding island. With the link and the full adjacency graph produced in box 30, all the objects of the island can be retrieved. This and similar problems are elaborated on in the following section.

4. Output data structure

During algorithm execution the encoded data have to be collected and inserted into a dynamic data structure. Setting aside for the moment the dynamic growth of these data, the final result of our previous example could be stored in a tabular fashion, as shown by Fig. 9.

The leading entry of the table of Fig. 9 contains the objects of the new set, which according to previous discussion should

contain "frame" as its first member. The outer border of this imaginary object is, of course, not known, which explains why most of the following entries are left blank. The column "Adjacent to" carries the adjacency graph information. In our running example, only the object A is adjacent to "frame." The following columns of the graph are reserved for islands completely circumscribed by the object.

For the object "frame" only one island always exists, namely, the total image itself. The first column of the island entries is called "link," and this is where notations from box 19 in Fig. 7 are inserted. In our case, the first (and only) object in the image to be encountered from "frame" is object A. The next column "x, y" is the starting point of the chain that is stored in the following column. However, since the island inside "frame" is the image itself, this information is already known and can be left blank. The special case of the object "frame" is also reflected in the flowchart in Fig. 6, boxes 10 and 10a.

Let us now look at the second row of the table. The object A is first defined by its outer border, which we assume is what we see in Fig. 5(b) resulting in chain0. (Actually, this chain of A's is not shown in Figs. 4 and 5 since it would have obscured the explanatory discussion at that time.) In the "Adjacent to" column the information "frame" is inserted by box 30, together with the information A in the above line of the table. The object A has two islands the chains of which are generated by box 13, and the first objects to be encountered after island following are B and C, respectively.

The rest of Fig. 9 should be self-explanatory. Now, during execution, the table of Fig. 9 grows vertically as new objects are found, and the table must therefore be allowed to expand dynamically. Several of the entries, especially the chain-coded borders, have vastly varying lengths. These are preferably stored in an augmentable list structure using conventional techniques.

The adjacency graph, stored in the column "Adjacent to," can, directly or indirectly, give answers to questions about the image topology. It directly answers the question: Which objects are adjacent to object A? But it could also be used for questions like: Which objects are enclosed by object A? The answer to this question can be retrieved by using the island information. The first island enclosed by A consists of object B and all objects directly or indirectly adjacent to B such that the indirect adjacency path does not pass through A itself. The second island enclosed by A is object C, which happens to have no adjacencies other than A itself.

If the questions about islands and their object consistency are expected to be frequent, the columns "link" in Fig. 9 could be augmented to "object set," as shown in Fig. 10.

NS	*OS*	*x, y*	Chain-coded border	Adjacent to	First island			Second island		
					link	*x, y*	border	link	*x, y*	border
frame	-	-	-	*A*	*A*	-	-	-	-	-
A	0	0, 0	chain0	"frame," *B, C, E*	*B*	1, 1	chain1	*C*	8, 3	chain3
B	1	1, 1	chain2	*A, D, E, F*	-	-	-	-	-	-
C	2	8, 3	chain4	*A*	-	-	-	-	-	-
D	0	2, 4	chain5	*B, E*	-	-	-	-	-	-
E	3	4, 4	chain6	*A, B, D, F*	-	-	-	-	-	-
F	2	2, 7	chain7	*B, E*	-	-	-	-	-	-

chain0: 00000000000333333333322222222221111111111
chain1: 00000333333333222221111111111
chain2: 00000333222233032321111111
chain3: 00332211
chain4: 00332211
chain5: 00332211
chain6: 00333333322222110300112101 1
chain7: 00332211

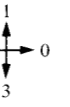

Figure 9 Output data.

First island			Second island		
object set	*x, y*	chain-coded border	object set	*x, y*	chain-coded border
A	-	-	-	-	-
B, D, E, F	1, 1	chain1	*C*	8, 3	chain3

Figure 10 Island consistency extracted from adjacency information in Fig. 9.

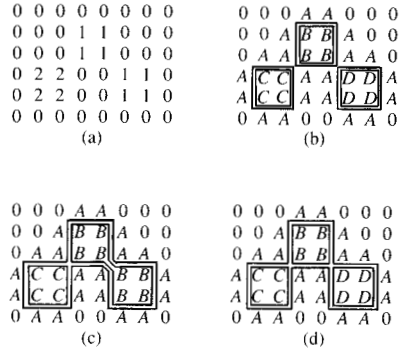

Figure 11 A nonbinary image and its three possible connectivity interpretations.

5. The 4/8-connectivity problem

The well-known problem of 4- *versus* 8-connectivity prevails in all cases where a Cartesian grid is used for digitization. In our case we can illuminate the problem with Fig. 11(a). Three equally valid topological interpretations are possible for this image:

1. The image consists of three islands, each one consisting of one single object. According to this interpretation the

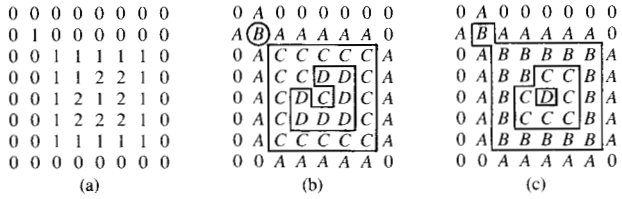

Figure 12 Another nonbinary image with connectivity interpretations as in Figs. 11(b) and (c), respectively.

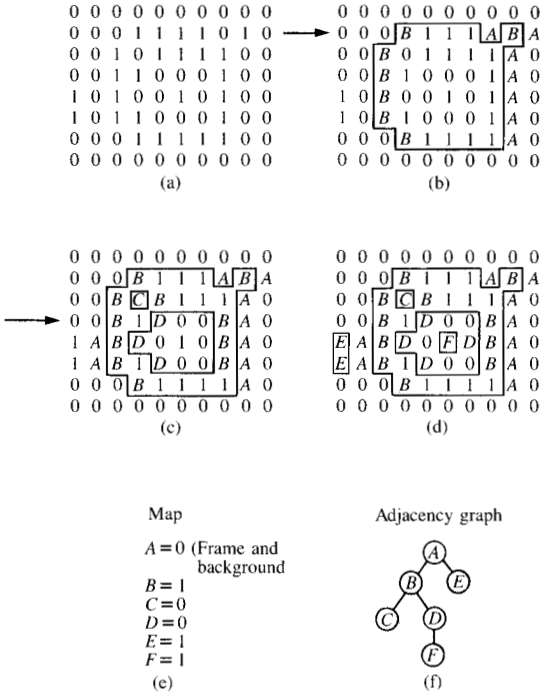

Figure 13 The new algorithm applied to a binary image.

segmentation algorithm should assign new labels, as shown by Fig. 11(b). The surrounding object is considered to be 8-connected during island following; the objects are considered to be 4-connected during object following.

2. The image consists of one island having two objects. According to this interpretation the algorithm should assign labels as shown by Fig. 11(c). This implies 4-connected island following and 8-connected object following.
3. The image consists of one island having three objects. According to this interpretation the algorithm should assign labels as shown by Fig. 11(d). This implies 4-connected island following and 4-connected object following.

In many ways the third alternative is very appealing, since it guarantees that the outer border of an object traced out during object following and the borders of possible islands inside the object traced out during island following have the same characteristics. All pixels of the object will be 4-connected. In the other two alternatives, there will be ambiguities, as illustrated by Fig. 12. The image of Fig. 12(a) will be translated by alternative 1 into Fig. 12(b) and translated by alternative 2 into Fig. 12(c). In both cases the object 1 with its two 8-connected pixels will be transformed into two objects and not into one (as if it were 8-connected) or into three (as if it were 4-connected). Obviously, alternative 3 would translate object 1 of Fig. 12(a) into three 4-connected objects (not shown in Fig. 12).

If we choose to use alternative 3 above, we have to make sure that another ambiguity does not arise. In Fig. 11 alternative 3 will produce only one island. The three objects must consequently be considered to be inside this island and adjacent to each other. If they are not noted as being adjacent by the algorithm, only the first object B will be retrieved as belonging to the island. This adjacency between, say, B and C must be noted during object following when this procedure has reached the upper right corner of object C, where it touches the lower left corner of object B. Here the crack link computation in box 23 (see Fig. 8) should make a clockwise sweep over the neighboring pixels. It will then find that the crack of the 4-connected object C makes a 90-degree downward turn. But it should also note that a new label $q = B$ is found, and the adjacency (B, C) should be inserted in the output data. This is something which is not included in the above flowchart description where all adjacency notations are made in box 30.

In summary, alternative 3 produces objects that consist of 4-connected pixels, but the objects themselves are considered 8-adjacent. Obviously, the different border-following procedures for the different alternatives could be given a more explicit and detailed description. They are omitted here for the sake of brevity.

6. A simplified version for binary images

Since a binary image is a special case of nonbinary images, it is possible to use the above algorithm as it is for segmentation of binary images. However, there is an inherent constraint in binary images that makes considerable simplification possible. The adjacency graph of a binary image takes the form of a tree, since an island in a binary image always consists of one object only (possibly with holes inside). Consequently, the tracing of the island and the tracing of the object contour no longer have to be two separate procedures, as in the nonbinary case. Furthermore, as soon as the raster scanner hits a boundary, there is only one adjacency notation to be made, namely, between the surrounding and the surrounded object, the island.

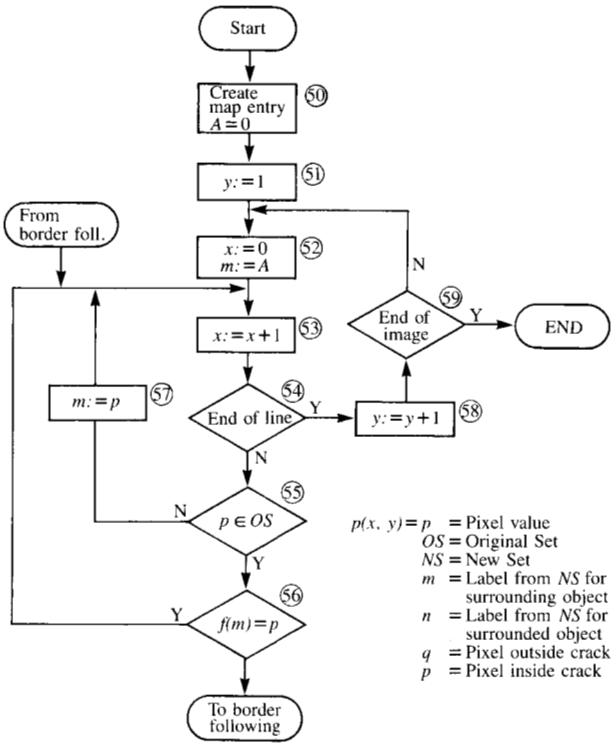

Figure 14 Raster scanning of binary images.

The simplified algorithm for the binary case is illustrated by the example in Fig. 13. Note that we only have to write new labels to the east of the cracks. This is enough to make the raster scanner aware of which boundaries have already been followed.

The flowcharts for raster scanning and border following are shown by Figs. 14 and 15, respectively, and should be rather self-explanatory. The 4/8-connectivity problem is solved by adopting different rules for 0-objects and 1-objects, as can be seen in Fig. 13. Here we have chosen to let the 0-objects consist of 4-connected pixels while the 1-objects are defined as sets of 8-connected pixels. This choice determines the details of the crack link computation in box 64 in Fig. 15.

The output data for the example of Fig. 13 are shown by Fig. 16. We assume that the imaginary frame around the original image of Fig. 13(a) consists of 0's. This frame and its background extension in the image itself is the object A. The adjacency information for each object consists of one upward link towards the root of the tree and one downward link to surrounded objects. The area of an object is defined by its outer contour and by the contour of its surrounded objects.

The generalization of the above binary algorithm so that it can handle "pseudo-binary" images exemplified by Fig. 2 is a trivial matter. Such nonbinary images have the same topological constraints as binary images where islands always consist of one object only. Consequently, the adja-

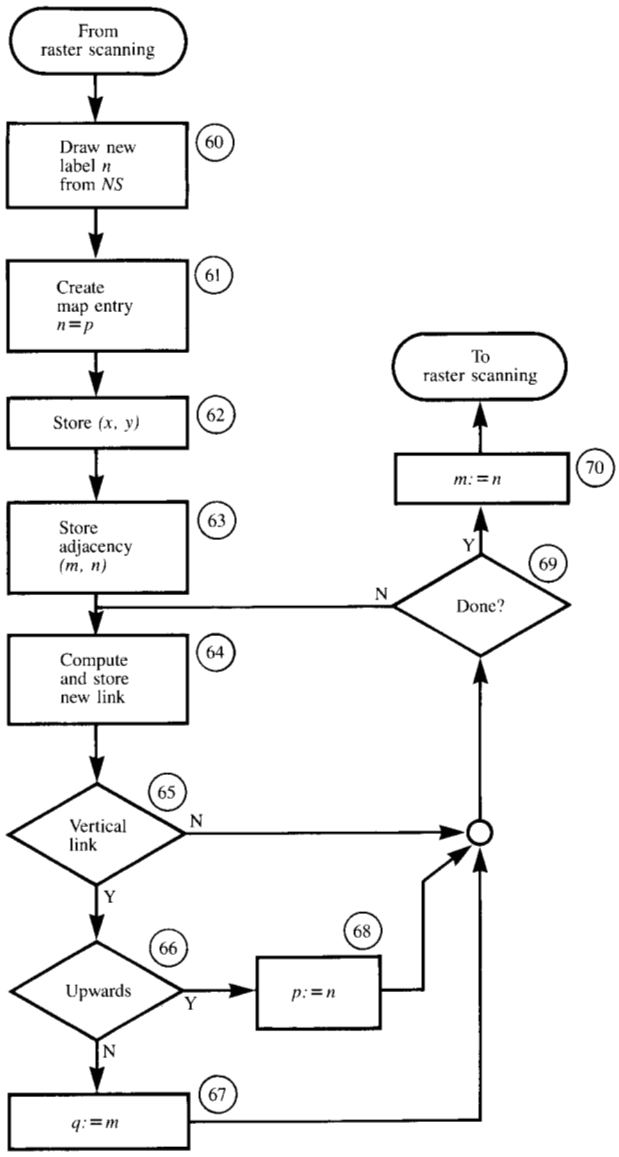

Figure 15 Border following of binary images.

Map		Adjacency		Outer contour	
NS	*OS*	Up	Down	*x*, *y*	Chain
A	0	-	*B*, *E*	-	-
B	1	*A*	*C*, *D*	3, 1	00003010323333322222211110
C	0	*B*	-	3, 2	0321
D	0	*B*	*F*	4, 3	00033322212101
E	1	*A*	-	0, 4	033211
F	1	*D*	-	5, 4	0321

Figure 16 Output data for the binary image of Fig. 13.

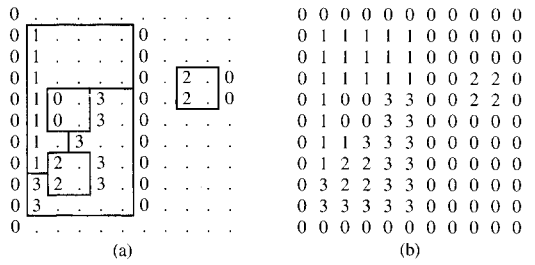

Figure 17 First step in the decoding procedure: border coating.

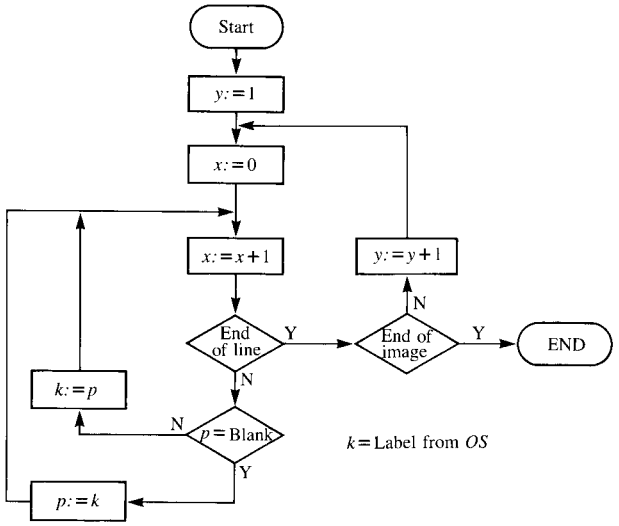

Figure 18 Flowchart for area filling.

cency graphs for pseudo-binary images are trees just as for binary images, and there is no need for separate island and object following. The 4/8-connectivity problem is most conveniently solved by mimicking the binary case and using 4-connectivity and 8-connectivity interchangeably for every second object level in the adjacency tree.

Finally, a comparison can be made with Kruse's algorithm [3]. The algorithm of Figs. 14 and 15 in this paper seems to be somewhat simpler and cleaner, since it requires no tagging. One bit per pixel is saved in the image memory. The stack principle of [5, 6] is not employed. The binary image algorithm of this paper has been explained briefly in [7].

7. Decoding

The output data from the above algorithms may eventually be used in several ways that fall outside the scope of this paper. But since the output data permit full reconstruction of the input data, something ought to be said about their use for decoding.

In decoding and reconstruction of the input image we do not use the adjacency information in Fig. 9, which makes things very simple. Initially, the image plane is filled by blanks (.) which take up one state of the available pixel states. Then, we do the reverse of border following, which is to use the available chains in the output data for laying out the contours. For the object border chains the pixel values from the old set {0, 1, 2, 3} are inserted in the image plane inside and to the east of each crack. Insertion of pixel values to the north, west, and south is not necessary.

For the island border chains the pixel values from the old set are inserted outside and to the east of the crack. When all borders are "coated" in this way on their eastern side, the data of Fig. 9 result in the image shown by Fig. 17(a). It is now a trivial matter for the raster scanner to do its area filling job, which results in Fig. 17(b). The flowchart for the area filling is given in Fig. 18.

8. Conclusions

The algorithm presented in this paper delivers a contour-coded version of a nonbinary image combined with a topological description of adjacency conditions. It also provides data that determine how islands of objects are completely surrounded by another object. To be able to do this, all border elements are traced out twice. The first time the contour of the object on the outside of the border is followed in a closed path. The second time the contour of the object on the inside of the border is followed in a closed path.

In the binary image case these two paths are identical and can be combined into one procedure. In the nonbinary image case they are only piecewise identical and these operations have to be done as two separate procedures.

During border following the labels closest to the border are given new labels. These new labels serve two purposes. They enable the algorithm to establish the wanted adjacency conditions, and they serve as indicators to the raster-scanning part of the algorithm, telling this procedure that this border is already done.

In the nonbinary case tagging is used to distinguish between a case in which a border is followed on one side and a case in which it is not followed at all. Actually, tagging could be avoided completely if all objects were known to have a width of two pixels or more.

Throughout the paper we have used 4-directional links to encode the contours. Such 2-bit links are sufficient to follow

a crack, the actual exact borderline between two regions. We feel that crack-coding is both a simpler and a more natural way to encode contours than the more prevailing method of establishing a chain that forms a path between pixels on one side or the other of the contour. However, the latter method of contour encoding can also be used in the presented algorithm.

The 4/8-connectivity problem for objects in a Cartesian grid can be resolved in at least three ways for the nonbinary case. The most consistent way seems to be to define all objects as internally 4-connected (between pixels) but externally 8-adjacent to each other.

For binary images the algorithm lends itself to considerable simplification.

Finally, reconstruction of the original image from the algorithm output data has been demonstrated to be simple and straightforward.

References

1. T. H. Morrin, "Chain-link Compression of Arbitrary Black-White Images," *Computer Graph. & Image Process.* **5,** 172–189 (June 1976).
2. R. Cederberg, "Chain-Link Coding and Segmentation for Raster Scan Devices," *Computer Graph. & Image Process.* **10,** 224–234 (1979).
3. R. Cederberg, "On the Coding, Processing and Display of Binary Images," Ph.D. Dissertation No. 57, Linkoping University, S-58183 Linkoping, Sweden.
4. P.-E. Danielsson, "Encoding of Binary Images by Raster-Chain-Coding of Cracks," *Internal Report Lith-ISY-I-0497,* Linkoping University, S-58183 Linkoping, Sweden; also in *Proc. 6th Int. Conf. on Pattern Recognition,* IEEE, Oct. 1982.
5. B. Kruse, "A Fast Algorithm for Segmentation of Connected Components in Binary Images," *Proceedings of First Scandinavian Conference on Image Analysis,* Studentlitteratur, Lund, Sweden, January 1980.
6. B. Kruse, "A Fast Stack-Based Algorithm for Region Extraction in Binary and Nonbinary Images," *Signal Processing: Theories and Applications,* M. Kunt and F. de Coulon, Eds. (Proceedings of EUSIPCO Conference, August, 1980), North-Holland Publishing Co., Amsterdam, 1980, pp. 169–173.
7. P.-E. Danielsson, "An Improvement of Kruse's Segmentation Algorithm," *Computer Graph. & Image Process.* **17,** 394–396 (1981).

Received August 17, 1981; revised February 2, 1982

Per-Erik Danielsson *Linköping University, S-581 83 Linköping, Sweden.* Professor Danielsson is a professor of computer engineering at Linköping University. His main interests are computer architecture and algorithms for image processing. Before joining the Linköping faculty in 1971, he was associated with the University of Lund, Research Institute for National Defense, Stockholm, and Saab Aircraft, Linköping. He has published a textbook on digital techniques. Professor Danielsson graduated from the Royal Institute of Technology in Stockholm and received degrees in 1960 and 1967. He received an honorary doctorate from Linköping University in 1982 for promoting computer science education. Professor Danielsson was a visiting scientist at the IBM Research Division laboratory in San Jose, California, during the academic year 1980 to 1981.