# Deriving Topological Representations from Edge Images

Ullrich Köthe

University of Hamburg, Cognitive Systems Group
Vogt-Kölln-Str. 30, 22527 Hamburg, Germany
koethe@informatik.uni-hamburg.de

**Abstract:** In order to guarantee consistent descriptions of image structure, it is desirable to base such descriptions on topological principles. Thus, we want to be able to derive topological representations from segmented images. This paper discusses two methods to achieve this goal by means of the recently introduced XPMaps. First, it improves an existing algorithm that derives topological representations from region images and crack edges, and second, it presents a new algorithm that can be applied to standard 8-connected edge images.

## 1 Introduction

Many authors have argued that image segmentation should produce a topological image representation [2, 9, 11, 17]. Otherwise, it would be impossible to consistently answer basic questions such as “Which regions are neighbors of each other?”, “Which regions are contained in another one?”, or “Where precisely is the mutual boundary between two neighboring regions?” The connectivity paradox is the most infamous example for the inconsistencies that occur if a naïve image representation is used [11].

A topological representation defines a finite topological space made up of regions, arcs, and points (also called 2-, 1-, and 0-cells, or faces, edges, and nodes) which encode a particular partitioning of the image plane. Several structures have been proposed to encode such a partitioning, including cellular complexes [11], combinatorial maps [7], and the Khalimsky grid [8]. In a recent paper, I’ve introduced the concept of an *extended planar map* (XPMap) that subsumes the important characteristics of these structures and provides a unified approach to topological image representation [10].

In this paper, I’d like to fill a gap that remained open in the previous article: How can one actually derive a topological representation from a set of pixels? Or, put differently: How can we modify well known segmentation algorithms so that they produce the desired XPMap representation? Previous authors, e.g. [2, 17] have approached this problem solely on the basis of *crack edges*, that is, edges that are located between the pixels. We will review this work below. However, many standard image segmentation algorithms (such as Canny’s algorithm and the watershed algorithm) do not locate edges between pixels, but mark edges on the pixels themselves. As far as I can tell, an algorithm to transform this kind of edge image into a topological representation without resorting to heuristics (as in [14, 15]) does not yet exist. The development of such an algorithm is the main contribution of this paper.

## 2 Finite Topological Spaces and Topological Transformations

### 2.1 XPMaps

From the point of view of finite topology, image segmentation is the partitioning of the image plane into points, arcs, and regions. In principle, these entities can be defined by geometric means: A point can be defined by its 2D coordinate. An arc is a mapping of the open interval (0, 1) onto the image plane, such that the images of 0 and 1 coincide with two of the points and no arc crosses another one or itself. And regions can be defined as the connected components of the complement of the *boundary set*, i.e. the complement of the union of points and arcs.

However, the topology of the partitioning is only implicitly represented by this geometric definition. In the context of image analysis, making the topology explicit is much more desirable. A separation between topology and geometry can be achieved by means of abstract topological representations such as the XPMaps introduced in [10]. XPMaps are based on *combinatorial maps* [7]:

**Definition 1:** A *combinatorial map* is a triple $(D, \sigma, \alpha)$ where $D$ is a set of *darts* (also known as *half-edges*), $\sigma$ is a permutation of the darts, and $\alpha$ is an involution (a permutation with cycle length 2) of the darts. A combinatorial map is said to be *trivial* if it doesn't contain any dart. By definition, the trivial map contains a single face, the *infinite* face (which corresponds to the entire plane).

The cycles or *orbits* of the $\alpha$ involution define the *edges* (1-cells) of the map, and the orbits of the $\sigma$ permutation are its *nodes* (0-cells). The mapping $\varphi = \sigma^{-1}\alpha$ is also a permutation whose orbits correspond to the *faces* (2-cells). A $k$-cell is said to *bound* an $m$-cell if $k < m$ and the orbits associated with either cell have at least one dart in common. A combinatorial map fulfills the axioms of a topological space, if open sets are defined as follows: a set $O$ of cells is *open* if, whenever cell $c$ belongs to $O$, all cells bound by $c$ do also belong to $O$.

A combinatorial map is *planar* (encodes a partitioning of the plane) if the numbers $n, e, f$ of nodes, edges and faces respectively fulfill *Euler's equation*:

$$n - e + f = 2 \qquad (1)$$

However, this equation only applies if the map's boundary set, i.e. the graph defined by considering only nodes and edges, is connected. Thus, combinatorial maps cannot represent partitionings of the plane with disconnected boundaries, such as a wall with windows. But disconnected boundaries are common in image analysis, so that the map concept must be extended in order to handle this case:

**Definition 2:** An *extended planar map* (*XPMap*) is a tuple $(C, c_0, \textit{exterior}, \textit{contains})$ where $C$ is a set of non-trivial planar combinatorial maps (the *components* of the XPMap), $c_0$ is a trivial map that represents the infinite face of the XPMap, *exterior* is a relation that labels one face of each component in $C$ as the exterior face, and *contains* is a relation that assigns each exterior face to exactly one non-exterior face or the infinite face.

Figure 1 shows an example of an XPMap. Details can be found in [10].

Components: $C_0$=Infinite, $C_1$, $C_2$

Darts: $C_1 = \{1, 1', 2, 2', \ldots, 5, 5'\}$ (black)
$C_2 = \{6, 6', 7, 7', 8, 8'\}$ (gray)

Edges: $E_1 = (1,1')$, $E_2 = (2,2')$, …, $E_8 = (8,8')$

Nodes: $N_1 = (1', 4)$,…, $N_4 = (3', 5, 4')$,
$N_5 = (5')$, …, $N_8 = (7', 8')$

Faces: Infinite, $F_1 = (1, 4, 5, 5', 3', 2')$,
$F_2 = (1', 2, 3, 4')$, $F_3 = (6', 7, 8')$, $F_4 = (6, 8, 7)$

Exterior: exterior($C_1$) = $F_1$, exterior($C_2$) = $F_3$

Contains: Infinite contains $F_1$, $F_2$ cont. $F_3$

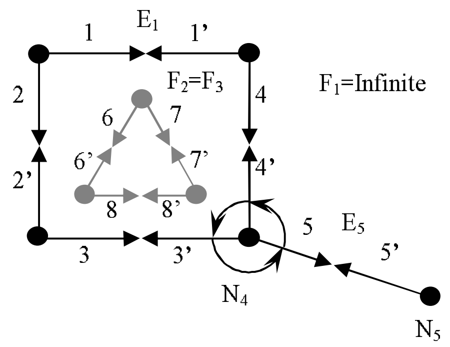


**Fig. 1:** Example of an XPMap (for clarity, only some edges and nodes are labeled in the drawing, and only the $\alpha$ orbits [opposite arrows] and the $\sigma$ orbit for node $N_4$ are shown).

Since an XPMap may contain multiple components, Euler's equation has to be modified to take this into account:

$$n - e + f - c = 1 \tag{2}$$

where $c$ denotes the number of components.

XPMaps can be modified by means of *Euler operators*. These operators are named so because they guarantee that Euler's equation remains valid after the modification. Thus, Euler operators are able to transform a valid XPMap into another valid XPMap. In the present context of image segmentation, four operators are of primary interest (see figure 2, detailed algorithms and proofs can be found in [10]):

MERGE FACES, REMOVE BRIDGE: These operators remove an edge. They first remove the edge's darts from their $\sigma$-orbits and then delete the edge's $\alpha$-orbit and the darts. The desired modification of the $\varphi$-orbits follows automatically. Removal of a bridge (an edge that bounds only one face) creates a new XPMap component, so the *exterior* and *contains* relations must be adjusted as well.

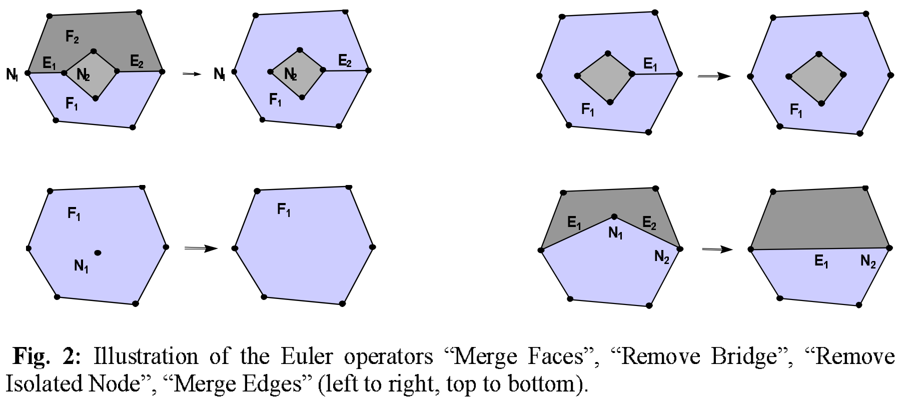


**Fig. 2:** Illustration of the Euler operators "Merge Faces", "Remove Bridge", "Remove Isolated Node", "Merge Edges" (left to right, top to bottom).

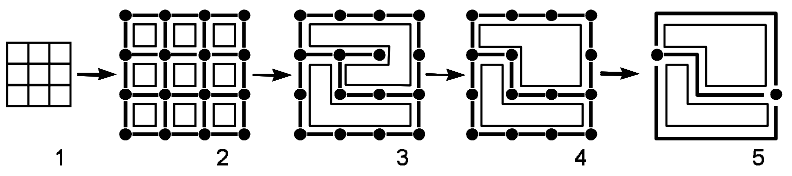


**Fig. 3**: Segmentation by Euler operators: 1) original image; 2) associated Khalimsky grid (one 2-cell per pixel); 3) after 7 applications of Merge Faces; 4) 1 application of Remove Bridge and Remove Isolated Node; 5) 13 applications of Merge Edges

REMOVE ISOLATED NODE: Sometimes the new component resulting from "Remove Bridge" consists of a single node. "Remove Isolated Node" deletes this component from the XPMap and updates the *contains* relations accordingly.

MERGE EDGES: Edge removal usually causes many nodes to have degree 2, i.e. to bound exactly two edges. These nodes are called *links*. "Merge Edges" simplifies an XPMap by replacing a connected sequence edge-link-edge with a single edge.

All these operators reduce the number of cells. To use them for segmentation, we must start with an over-segmentation which is then successively reduced to the desired segmentation.

## 2.2 Creating a Topological Partitioning by Euler Operators

The simplest way to define a topologically consistent over-segmentation is to associate a face with *every* pixel. This is formalized by means of the *Khalimsky grid* [8]:

**Definition 3:** A *Khalimsky grid* is defined on $\mathbb{Z}^2$ by denoting points with two even coordinates as *faces*, points with two odd coordinates as *nodes*, and mixed points as *edges*. Nodes bound their eight neighbors (four edges and four faces), and edges bound the two neighboring faces.

If open sets are defined as in the previous section Khalimsky grids also fulfill the axioms of a topological space. It is also easy to see that a Khalimsky grid defines a combinatorial map: we associate two darts with every edge, one pointing in increasing $x$ or $y$ direction respectively, the other pointing into the opposite direction. These pairs form the orbits of the $\alpha$ involution. The orbits of the $\sigma$ permutation are defined by taking the four darts starting at the same node and sorting them in mathematically positive order. For any image there exists a corresponding Khalimsky grid such that the pixel at image coordinate $(x, y)$ corresponds to the face at Khalimsky coordinate $(2x, 2y)$.

Figure 3 illustrates how an image segmentation can be obtained by applying a sequence of Euler operators to a Khalimsky grid. However, as images get larger, this method becomes inefficient. Therefore, we will not pursue this approach in the present paper. Instead, we will investigate two alternative algorithms that derive topological representations directly from traditional image segmentations.

## 3 The Crack Insertion Algorithm

Topological segmentation on the basis of Euler operators, as outlined in the previous sections, is a good way to theoretically prove the topological properties of the resulting representations. However, it is somewhat laborious as a practical algorithm. In practice, the *Crack Insertion Algorithm* is the easiest way to derive a topological representation. In similar form, phase 1 of the algorithm has been used in the segmentation methods of [2, 3, 17]. However, phase 2 (derivation of an explicit topological representation) was only carried out in [2].

The algorithm starts from a region image, i.e. a complete image partitioning into 4-connected components. Region images typically result from region growing (e.g. [1]), split-and-merge (e.g. [12]) or 4-connected components labeling of binary images. As the name suggests, crack edges are located *between* neighboring pixels, if those pixels' labels differ. In a region image, the crack edges are only coded implicitly because the image does not contain addressable entities which represent the cracks. The crack insertion algorithm makes the cracks explicit by inserting appropriate entities into the data representation (phase 1). After this an XPMap representation can easily be derived (phase 2). Compare figure 4 for illustration:

**Phase 1: Crack Insertion**

Given: Region image with labeled 4-connected components, size $w \times h$.

1. Create an image of size $(2w - 1) \times (2h - 1)$. Following [3], we will call this image the *super grid*. Copy the labels from position $(x, y)$ of the region image to position $(2x, 2y)$ in the super grid.
2. For each cell in the super grid with coordinates $(2m, 2n + 1)$: If the cell's two vertical neighbors have the same label, copy this label into the cell. Otherwise, mark the cell with a special *edge* label.
3. For each cell in the super grid with coordinates $(2m + 1, 2n)$: If the cell's two horizontal neighbors have the same label, copy this label into the cell. Otherwise, mark the cell with a special *edge* label.
4. For each cell in the super grid with coordinates $(2m+1, 2n+1)$: If any of the 4-neighbors was labeled as an edge, mark the cell with a special *node* label. Otherwise, copy the label of the neighbors (which is necessarily unique).

Phase 1 results in 4-connected regions whose cells have identical labels, and 4-connected boundaries, whose cells are labeled as edges and nodes (the proof of 4-connectedness is straightforward). Now, phase 2 derives an XPMap from the labeled super grid:

**Phase 2: Derivation of an XPMap from a Labeled Super Grid**

Given: Labeled super grid resulting from phase 1.

1. Augment the super grid with a one cell wide border whose cells are labeled as nodes if their coordinates are both odd, and as edges otherwise. This results in a super grid of size $(2w + 1) \times (2h + 1)$.
2. Find the connected components of the boundary set (that is, of the set of cells marked as edges and nodes). For each component thus found create a component in the XPMap to be build.

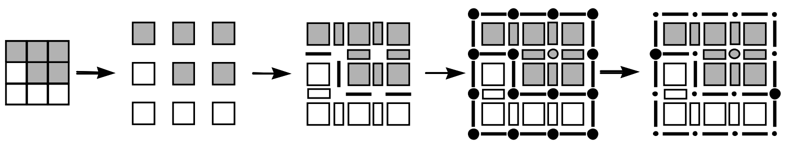


**Fig. 4:** Application of the Crack Insertion Algorithm: 1) 3×3 region image; 2) Labeling of the even coordinate pixels in the associated super grid; 3) Labeling of the mixed coordinate pixels (the special edge label is indicated by a black line); 4) Labeling of the odd coordinate pixels and addition of the outer boundary (the special node label is indicated by a black ball); 5) Elimination of joints by "Merge Edges". Note that, in contrast to figure 3, Euler operators cannot eliminate pixels from a super grid. Instead, pixels get re-labeled with their new cell type. In this case, black balls (nodes) are changed into black dots (odd coordinate pixels that belong to an edge).

3. For each component:
   - 3.1. For each edge in the component: Create a pair of darts and insert it as an orbit into the component's $\alpha$ involution. The two darts of a pair are distinguished by their orientation (north/south and east/west respectively).
   - 3.2. For each node in the component: Create an orbit in the component's $\sigma$ permutation that contains the darts adjacent to the node in counter-clockwise order.
4. Establish the *exterior* relation of each component. In case of the outermost component (the one that is adjacent to the super grid's border) the exterior face is always the infinite face (which is not explicitly represented in the super grid). To find the exterior face of the other components, traverse the super grid in scan-line order. When the scan first encounters a node of a particular component, the cell seen just before belongs to the sought for exterior face.
5. Establish the *contains* relation as follows: for each region, list the components where the present region was designated as the exterior face.

Since $\varphi = \sigma^{-1}\alpha$ it is not necessary to explicitly derive the $\varphi$ permutation from the super grid – it is already uniquely defined by $\sigma$ and $\alpha$. It should be noted that the algorithm is essentially equivalent to a segmentation by means of Euler operators, as outlined in the last section: If we started with a Khalimsky grid on the subset $[-1, w] \times [-1, h]$ of $\mathbb{Z}^2$, the edges and nodes to be removed by Euler operators would be precisely the ones that were not marked with edge/node labels during crack insertion (compare figures 3 and 4).

After phase 2 of the above algorithm, all edges consist of a single cell, and most nodes bound exactly two edges. We will call these nodes *joints*, whereas nodes of higher degree will be called *junctions*. In many applications, we are only interested in the junctions, not the joints. In this case, we may again use the Euler operator "Merge Edges" to transform connected cell sequences of the form "edge-joint-edge" into single edge cells. By applying the operator repeatedly, we can successively eliminate all joints (last step in figure 4). Details can again be found in [10].

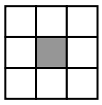
## 4 Deriving an XPMap from an 8-Connected Boundary

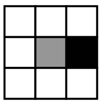
Many segmentation algorithms do not present their results by means of region images or crack edges but rather mark some pixels as "edge pixels". This kind of edge image typically results from the watershed algorithm [16] and from topological thinning [6]. Canny's algorithm [4] is also commonly used to create edge images, although one can also directly produce a symbolic representation, e.g. a set of edgel chains (but in contrast to our new algorithm, existing algorithms to link edgels into chains heavily rely on heuristics, especially at junctions, e.g. [14, 15]). Edge images resulting from zero-crossing detection [15] occupy a middle ground: Since zero-crossings occur between pixels, it is natural to interpret them as crack edges, so that the crack insertion algorithm can be applied. But it is just as common to mark the pixel nearest to a zero-crossing as an edge pixel, in which case the resulting edges can be treated like Canny edges. As far as I'm aware of, a non-heuristic algorithm that can derive an XPMap or another topological representation from an edge image has not yet been developed. The new algorithm will be based on the following observations:

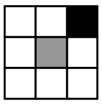
- From the definition of the XPMap, it is clear that we need not only regions and edges, but also nodes (junctions and end points). Thus, a classification of the boundary pixels into edge and node pixels will be the core of our algorithm. Considering this, it is not really correct to call the images resulting from Canny's algorithm "edge images" because this ignores the nodes. In the sequel, we use the term *boundary image* instead.
- It is well known that one cannot in general define consistent topological relations on an image by using either the 4- or 8-neighborhood. However, one can use 4-neighborhood in the foreground, and 8-neighborhood in the background [13]. Therefore, we will adopt the 4-neighborhood to determine the connectivity of regions and the 8-neighborhood for the connectivity of the boundary.
- We will define edges as junction-free *chains* of edge pixels. To be junction-free, chains must have the following property: Every interior pixel of the chain must be 8-connected to exactly two other edge pixels. The two ends of the chain are adjacent to an edge pixel and a node pixel. A *degenerate chain* consists of a single edge pixel that is adjacent to two nodes.
- We can determine whether a boundary pixel is a node or edge pixel by just looking at the 8-neighborhood of the pixel[1]. However, this requires the boundary to be *irreducible*: It must not contain *simple points*, i.e. boundary pixels that could be turned into region pixels without changing the connectivity of both the boundary and the regions. Removal of simple points is called *thinning* (therefore, we also call an irreducible boundary *thin*).

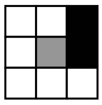
  The watershed algorithm is an example of thinning, where a cost function (e.g. the image gradient) determines the order of simple point removal. Therefore, boundary images resulting from this algorithm fulfill the requirement. In contrast, boundaries coming from Canny's algorithm may still contain a few simple points which must be removed before the classification can start.

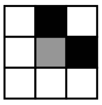
[1] Formally, the 8-neighborhood of the pixel at $(x_0, y_0)$ is defined by $N_8(x_0, y_0) = \{(x, y)\colon \max(|x-x_0|, |y-y_0|) = 1\}$, i.e. the center pixel is not part of the neighborhood.

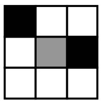

1 (*)
node pixel

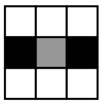

2 (*)
node pixel
or reducible

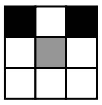

3 (*)
node pixel
or reducible

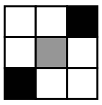

4
cannot occur
(not thin)

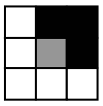

5 (*)
node pixel
or reducible

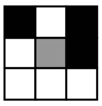

6
edge pixel

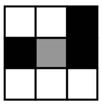

7
edge pixel

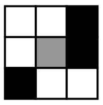

8
edge pixel

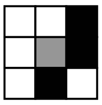

9
edge pixel

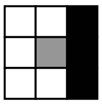

10
cannot occur
(not thin)

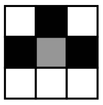

11
edge pixel

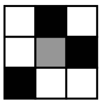

12
edge pixel

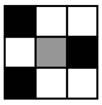

13
edge pixel

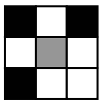

14
cannot occur
(not thin)

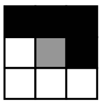

15
cannot occur
(not thin)

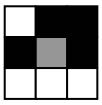

16 (*)
node pixel
or reducible

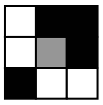

17
node pixel

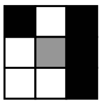

18
node pixel

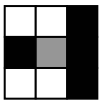

19
node pixel

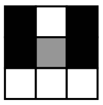

20
cannot occur
(not thin)

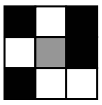

21 (*)
node pixel
or reducible

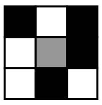

22
node pixel

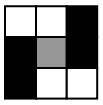

23
edge pixel

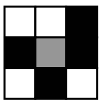

24
edge pixel

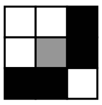

25
edge pixel

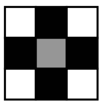

26
node pixel

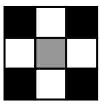

27
node pixel

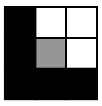

28
edge pixel


29 (*)
node pixel
or reducible


30
cannot occur
(not thin)


31
node pixel


32
node pixel


33
cannot occur
(not thin)

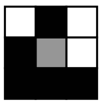
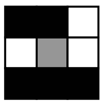
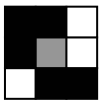
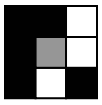
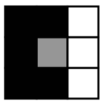
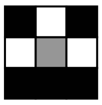
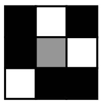
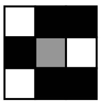
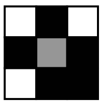
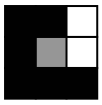
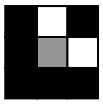
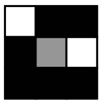
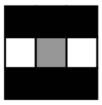
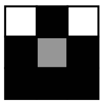
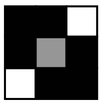
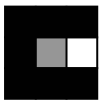
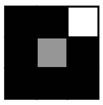
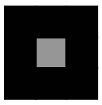
|  |  |  |  |  |  |
|---|---|---|---|---|---|
|  | 34 (*)<br>node pixel<br>or reducible |  | 35<br>edge pixel |  | 36<br>cannot occur<br>(not thin) |
|  | 37<br>node pixel |  | 38<br>cannot occur<br>(not thin) |  | 39<br>node pixel |
|  | 40<br>node pixel |  | 41<br>cannot occur<br>(not thin) |  | 42<br>node pixel |
|  | 43<br>cannot occur<br>(not thin) |  | 44<br>node pixel |  | 45<br>cannot occur<br>(not thin) |
|  | 46<br>edge pixel |  | 47<br>node pixel |  | 48<br>node pixel |
|  | 49<br>cannot occur<br>(not thin) |  | 50<br>node pixel |  | 51<br>node pixel |

**Table 1:** Possible configurations (modulo rotation and reflection) in the 8-neighborhood of a boundary pixel, along with the classification according to definition 4. Configurations marked with (*) can only occur if they are treated specially during thinning, and are then classified as “node pixels” (see text).

When we analyze how region and boundary pixels can be distributed in the 8-neighborhood, we obviously arrive at exactly 256 possible configurations. After removing rotated and reflected patterns, 51 unique configurations remain. They are shown in table 1. A number of these patterns cannot occur in an irreducible boundary because the center pixel would be a simple point[2]. In a few cases (marked with *) the decision is not clear cut: In a strict sense these patterns are reducible, but it is often desirable to modify the thinning algorithm in order to keep the points in the boundary.

Consider, for example, configuration 5: This pattern occurs at the corners of an axes-parallel rectangle. Removal of the center point would “round” the corners. A similar situation is found in configuration 16 (and 21, 29, 34): These configurations mark T-junctions. Removal of the center point would result in a little bend in an otherwise straight edge. Configurations 2 and 3 represent another exceptional case:

[2] On first glance one might think that even more patterns should be reducible, e.g. numbers 22, 37 and 51. But, as figure 5 shows, this is not the case: there are irreducible configurations that contain these patterns.

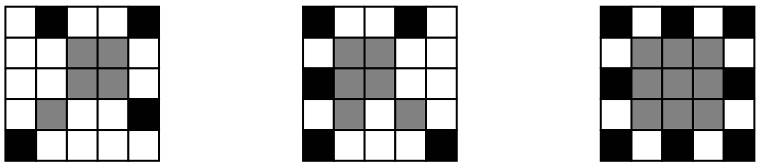


**Fig. 5:** Some patterns which show that configurations 22, 37, and 51 (located in the center of the example images, respectively) can actually by irreducible.

These patterns mark an end of a broken or dangling edge. Broken edges often result from noise or low contrast and can be "repaired" by higher level analysis and perceptual grouping. But this is only possible if the dangling ends are not removed during thinning. Since our algorithm does not depend on whether these configurations occur or not, there is no need to forbid them – the appropriate variant of thinning can be chosen according to the application context.

In order to avoid special treatment at the image border (where the 8-neighborhood would be partially outside the image) it is useful to again augment the boundary image with a one pixel wide border whose pixels are all marked as boundary pixels. If addition of this boundary creates reducible pixels they should be removed by an additional thinning iteration. Then, if the 8-neighborhood is partially outside the enlarged image the missing pixels can always be considered as region pixels (namely as part of the infinite region), and classification proceeds as usual.

Formally, the irreducible patterns are classified as follows (see table 1):

**Definition 4:** A boundary pixel in a thin boundary image is classified as an edge pixel if its 8-neighborhood consists of exactly four 4-connected components, and neither of the components consisting of boundary pixels contains more than one 4-neighbor. Otherwise, the pixel is a node pixel. (If configuration 5 is allowed, it is treated exceptionally and marked as a node pixel as well.)

The first condition ensures that each edge pixel has exactly two neighbors, so that we can actually group edge pixels into chains. The second condition is necessary to avoid that no pixel is classified as a node pixel in configurations like figure 5 left. On the basis of the definition, we can specify our algorithm as follows:

**Algorithm: XPMap from 8-connected Irreducible Boundary:**

Given: Boundary image with irreducible 8-connected boundary and 4-connected regions (as indicated above, several thinning variants might be used).

1. Augment the image with a one pixel wide border whose pixels are all marked as boundary pixels.
2. Find the 8-connected components of the *entire boundary*, i.e. the set of all boundary pixels. Create an XPMap component for each boundary component.
3. Classify boundary pixels according to definition 4.
4. Perform 8-connected components labeling of the *node pixels*. If necessary reclassify pixels to make the components simply connected (see remark A below). Each resulting component becomes a node of the XPMap to be build.

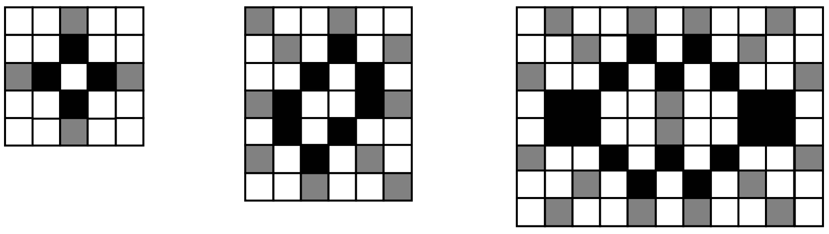


**Fig. 6:** Some patterns which result in connected components of node pixels (black) that have holes and are thus not simply connected.

5. Perform 8-connected components labeling of the *edge pixels*. In order not to merge different chains, a slight modification of the labeling algorithm is required: edge pixels that are adjacent to the same node pixel are not considered connected (see remark B below). Each resulting component is a chain and becomes an edge of the XPMap to be build.
6. For each XPMap component found in step 2:
   6.1. For each chain in the component: Create a pair of darts and insert it as an orbit into the component's $\alpha$ involution. Most chains have two unique end pixels, and the darts may be identified by those pixels. In case of a degenerated, one pixel chain (e.g. configuration 46), the darts are identified by their orientation. In case of a closed loop without node pixel (e.g. the boundary of a diagonal square), an arbitrary edge pixel in the chain must be re-classified as a node pixel.

   6.2. For each node in the component: Create an orbit in the component's $\sigma$ permutation that contains the darts adjacent to the node in counterclockwise order. To find those darts, simply walk around the node by means of the well-known "left hand on the wall" algorithm and register all darts thus met.
7. Establish the *exterior* and *contains* relations as in steps 4 and 5 of phase 2 of the crack insertion algorithm.

Remarks:

A. The nodes in an XPMap must be homeomorphic to a point. This means that the node components resulting from step 4 of the algorithm must be simply connected. Unfortunately, this is not guaranteed: it is possible to construct point configurations that result in connected components of node pixels that have holes. Figure 6 shows 3 examples. However, the problem is not very serious in practice since it can only arise if the boundary patterns are highly symmetric, and this is very unlikely in real images. In fact, the only configuration I have ever seen during experiments is figure 6 left which may occur if a seed has never got the opportunity to grow.

   In any case, the problem is easy to detect and surmount: measure the area enclosed by the node (i.e. the number of the node's pixels plus the area of a possibly enclosed hole) by means of the expression $A = \sum_i (x_i\, y_{i+1} - y_i\, x_{i+1})$ which is evaluated along the outer crack edge of the node. Compare the result-

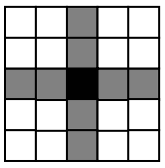


**Fig. 7:** Example of the problematic case where diagonally adjacent edge pixels (gray) touch the same vertex pixel (black). The four "arms" of the pattern are not considered connected at the center pixel, despite their touching diagonally.

ing area with the number of the node's node pixels. If the numbers agree, the node doesn't have a hole. Otherwise merge all pixels inside the hole with the surrounding node. In the practically important case of figure 6 left this simply means to re-label the central pixel, otherwise it amounts to a standard flood-fill. Nodes that occupy a simply connected image region rather than a single pixel present no topological problems, since the bounding relation (which determines the topological structure) is independent of a node's shape.

B. The edges in an XPMap must be homeomorphic to a line. This idea is captured by our notion of a *chain*: Starting from an edge pixel adjacent to a node, we must be able to go to uniquely defined successor pixels, until we reach another edge pixel which is adjacent to a node. We will now prove that the modified 8-connected component algorithm in step 5 indeed creates such chains.

First, we refer to figure 7 to justify the modification of the connected components algorithm: If two or more edge pixels are horizontally and vertically adjacent to a node pixel, they are also diagonally adjacent to each other. However, they clearly belong to different chains, and the modification explicitly handles this case in the desirable way.

Now we show that the labeling indeed produces chains. According to definition 4, the neighborhood of an edge pixel consists of four 4-connected components, two of which contain boundary pixels. We will call the latter *boundary groups*. If we look at the configurations classified as edge pixels in table 1, we see that none of their neighboring boundary groups consists of more than three boundary pixels. It is also easy to see that the chain property could only be violated if some boundary group would contain two or three edge pixels but no node pixel – only then the present edge pixel would not have a unique successor (or predecessor). But this is never the case: Whenever a boundary group consists of two or three pixels, at least one of them gets classified as a node pixel, so that the center pixel becomes the end of a chain. This fact is illustrated in figure 8. In other words, whenever an edge pixel is not the end of a

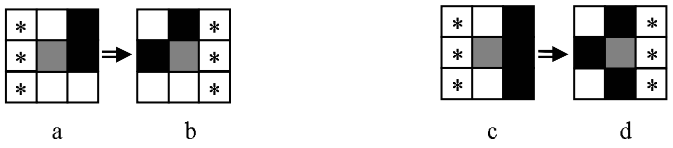


**Fig. 8: a, c:** The 2 possible edge pixel configurations with neighbor groups having 2 or 3 boundary pixels (* denotes an arbitrary attribution, a: configurations 11, 12, 13, 25, 28, 35; c: 23, 24, 35, 46). **b, d:** Configurations obtained by shifting the window 1 pixel to the right. As is easily verified by table 1, in any irreducible configuration containing these patterns the center pixel is classified as a node pixel (b: 5, 16, 17, 21, 27, 29, 34; d: 16, 29, 31, 42, 47).

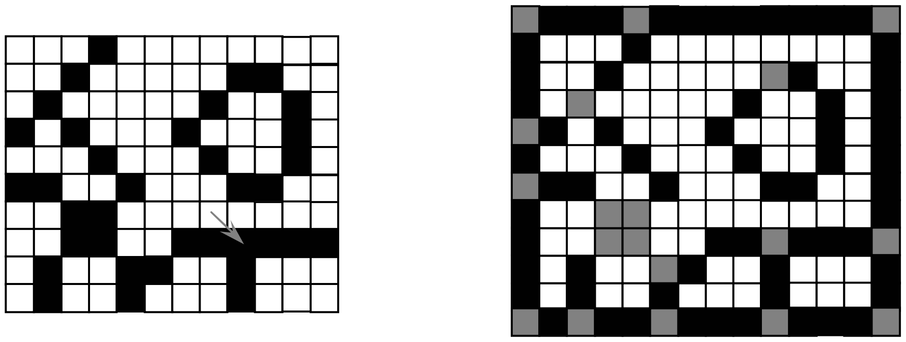


**Fig. 9:** left: boundary image as might result from the watershed algorithm (but note the modified thinning indicated with an arrow); right: resulting classification of the boundary pixels (black: edge pixels, gray: node pixels; note the addition of an outer boundary)

chain, its two neighboring boundary groups consist of exactly one boundary pixel, and the chain property is always ensured.

Although the description of the algorithm is somewhat complicated, its actual implementation is quite simple – it basically doesn't involve anything beyond connected components labeling (with slight modifications), contour following around nodes, and classification of boundary pixels according to the 8-neighborhood. The algorithm has been implemented successfully, and figure 9 shows a result.

## 5 Conclusions

This paper presented two algorithms that derive a topological representation from the results of standard segmentation algorithms: region images and edge images. This is very useful because it allows to apply topological concepts without major modifications to the segmentation algorithms themselves. Questions concerning the boundaries and neighborhood of features can thus be answered consistently, without resorting to heuristics that work around topological problems found in traditional representations.

It is easy to augment the new topological representation with geometric data defining the precise location of the nodes and edges. In fact, the clean separation of topology from geometry in the new framework provides very high flexibility, because different geometric models (e.g. straight lines, splines, sub-pixel accurate edgel chains) can be connected with the topological representation as the task requires. In contrast to topological representations used in computational geometry (e.g. [5, 7]), our approach establishes precise correspondences between the topological cells and the underlying raw pixels. Thus, one can always go back to the original pixel data when it becomes necessary to collect additional information about a cell's properties.

Further research should systematically compare the two approaches (crack edges or 8-connected edges) in order to define their appropriate application domains. For example, while the crack insertion algorithm is simpler, it also requires four times as many pixels in order to store the inserted cells, unless specific efficient data structures are used [2]. This problem doesn't arise with 8-connected edges. Also, the boundaries resulting from 8-connected edges look better visually. But making this statement objective is difficult because it is far from clear how segmentations should be compared. In fact, topological comparison criteria have rarely been used in the past. The presented results open up interesting new roads in this direction.

## 6 References

[1] R. Adams, L. Bischof: *"Seeded Region Growing"*, IEEE Trans. Pattern Analysis and Machine Intelligence, 16(6), pp. 641-647, 1994

[2] J.-P. Braquelaire, J.-P. Domenger: *"Representation of Segmented Images with Discrete Geometric Maps"*, Image and Vision Computing, 17(10),715-735, 1999

[3] C. Brice, C. Fennema: *"Scene Analysis Using Regions"*, Artificial Intelligence, 1(3), pp. 205-226, 1970

[4] J. Canny: *"A Computational Approach to Edge Detection"*, IEEE Trans. Pattern Analysis and Machine Intelligence, 8(6), pp. 679-698, 1986

[5] *"CGAL – Computational Geometry Algorithms Library"*, http://www.cgal.org/, 2002

[6] M. Couprie, G. Bertrand: *"Topological Grayscale Watershed Transformation"*, in: Proc. of SPIE Vision Geometry V, SPIE vol. 3168, pp. 136-146, 1997

[7] J.-F. Dufourd, F. Puitg: *"Functional specification and prototyping with oriented combinatorial maps"*, Computational Geometry 16 (2000) 129-156

[8] E. Khalimsky, R. Kopperman, P. Meyer: *"Computer Graphics and Connected Topologies on Finite Ordered Sets"*, J. Topology and its Applications, vol. 36, pp. 1-27, 1990

[9] U. Köthe: *"Generische Programmierung für die Bildverarbeitung"*, PhD thesis, Computer Science Department, University of Hamburg, 2000

[10] U. Köthe: *"XPMaps and Topological Segmentation - a Unified Approach to Finite Topologies in the Plane"*, in: A. Braquelaire, J.-O. Lachaud, A. Vialard (eds.): Proc. of 10th Intl. Conf. Discrete Geometry for Computer Imagery (DGCI 2002), Lecture Notes in Computer Science 2310, pp. 22-33, Berlin: Springer, 2002;

longer version appeared as: Univ. Hamburg, Dept. of Informatics Technical Report FBI-HH-M-308/0, 2001

[11] V. Kovalevsky: *"Finite Topology as Applied to Image Analysis"*, Computer Vision, Graphics, and Image Processing, 46(2), pp. 141-161, 1989

[12] T. Pavlidis: *"Structural Pattern Recognition"*, New York: Springer, 1977

[13] A. Rosenfeld: *"Adjacency in Digital Pictures"*, Information and Control vol. 26, pp. 24-33, 1974

[14] C. Rothwell, J. Mundy, W. Hoffman, V.-D. Nguyen: *"Driving Vision By Topology"*, in: IEEE Intl. Symposium on Computer Vision, pp. 395-400, 1995

[15] M. Sonka, V. Hlavac, R. Boyle: *"Image processing, Analysis, and Machine Vision"*, Brooks/Cole Publishing Comp., 1998

[16] L. Vincent, P. Soille: *"Watersheds in digital spaces: an efficient algorithm based on immersion simulations"*, IEEE Trans. Pattern Analysis and Machine Intelligence, 13(6), pp. 583-598, 1991

[17] S. Winter: *"Topological Relations between Discrete Regions"*, in: M. Egenhofer, J. Herring (eds.): Advances in Spatial Databases, pp. 310-327, Lecture Notes in Computer Science vol. 951, Berlin: Springer, 1995