COMPUTER VISION AND IMAGE UNDERSTANDING
Vol. 65, No. 1, January, pp. 38–56, 1997
ARTICLE NO. IV960484

Adaptive Vectorization of Line Drawing Images

RIK D. T. JANSSEN AND ALBERT M. VOSSEPOEL

Delft University of Technology, Faculty of Applied Physics, Pattern Recognition Group, Lorentzweg 1, 2600 GA Delft, The Netherlands

Received April 13, 1994; accepted September 22, 1995

A novel method for vectorizing line drawing images is presented. The method is based on a sequence of a standard vectorization algorithm and maximum threshold morphology, which can be iterated until a fitting criterion is met. First, ''anchor points'' are found in a coarse vectorization. The position of each anchor point is corrected by morphological operations, after which it is fixed. Next, the vectorization is refined between the anchor points. The method is adaptive because the original input image is used for correcting and improving the vectorization. Postprocessing modules can be added to anticipate specific properties of the line drawings to be vectorized. The vectorization method is evaluated using line drawings differing in scale and resolution. © 1997 Academic Press

1. INTRODUCTION

Vectorization is an often used approach for both reduction of data and interpretation of line drawing images [1–3]. It can also be used for constructing mosaics from separately scanned line drawings [4]. The vector representation has some distinct advantages over encodings by chain-elements, run-lengths, or skeletons. It is a sensible solution because the ''lines'' in the original scanned image (which are just adjacent pixels with no other mutual relation) are represented as geometric lines which can be used for shape representation or feature extraction. Another advantage is that the vectorization of an image is scale independent.

The reduction of pixels in a scanned image to lines with start and end coordinates is a nontrivial process due to the thickness of the lines, the angles of the lines, possible corruption of the input image (''missing'' object pixels, etc.), and the size of the images. For an A0 sized image (1 m^2) hundreds of megabytes are involved.

This paper presents an adaptive vectorization algorithm consisting of three parts (modules): an initial coarse vectorization, followed by determination of ''anchor points,'' and finally revectorization between these anchor points. The results can be improved by adding an adaptive clean up part.

Anchor points are so called because of two aspects: first, they are fixed at a certain stage of the algorithm, and second, they capture the gross structure of the lines in the original image. They describe the end points, corners, junctions, and other special points in the image, i.e. the obviously important (dominant) points of a vectorization.

The algorithm has been developed in the context of a map interpretation system. Therefore, examples are illustrated with this kind of line drawing. Of course, the algorithm can be applied to other types of line drawings such as CAD/CAM drawings, schematic diagrams, and lines extracted from an image with an edge detector. Indeed, one of the experiments is with an electrical diagram. There is no significant difference between the performance of the algorithm on maps or on diagrams. Between the different modules in the algorithm there are possibilities to anticipate specific properties of the line drawings.

This paper is divided into five parts. First, a classification of vectorization methods is described. Then, a short overview of published approaches to the automatic vectorization problem is given. This is followed by a description of our algorithm. Possible extensions are described and evaluated. Results, robustness, and improvements are discussed.

Paradigm

This paper focuses on approximating lines in a scanned image by straight line segments, called *edges.* Edges are specified by their start and end points, called *nodes,* and stored in a special data structure which topologically represents the graph. We explicitly chose *not* to use curve segments for approximating curves, because approximating with curve segments is even more difficult than approximating with lines. Curves are represented with sequences of short straight lines.

2. CLASSIFICATION OF VECTORIZATION METHODS

Converting line drawings from paper or film to a computerized representation can be performed in a number of ways. Lammerts van Bueren [5] considers the following methods.

1077-3142/97 $25.00

''Blind'' Vectorization

In this method, the drawing to be vectorized is pasted on a digitizing table. The operator uses a mouse for entering the start and end points of each line by clicking on them. He (she) is concentrated on the drawing and has no direct feedback on the result.

''Interactive'' Vectorization

The above procedure is used, but the operator is given feedback by displaying entered points and lines on a high resolution computer screen. The disadvantage is that the operator constantly has to move his head between the screen and the digitizing table. This may not be desirable from a human engineering point of view.

Scanning and ''Heads-Up'' Vectorization

After scanning, the map is displayed on a high resolution screen and vectorization is done manually using a mouse as above. The difference is that the result is displayed in an overlay on the map, giving a better opportunity for error detection. Some additional features, such as ''snapping'' of points on a grid, may be available in such a system.

Scanning and ''Interactive'' Vectorization

After scanning, software and an operator are necessary for the vectorization process. When the software is not able to make a ''good'' decision as how to go on with the vectorization, the judgement of the operator is asked. The advantage is that some vectorization is performed automatically, the disadvantage is that—depending on whatever one considers ''good''—numerous parameters must be chosen in advance to obtain a satisfactory vectorization (there is no uniform procedure for each type of map).

Scanning and Automatic Vectorization

After scanning, the map is vectorized automatically. This method is sometimes called ''batch vectorization.'' This is what we aim for in this paper: in ambiguous situations, the system has to make decisions independent of an operator. This independence calls for the optimization of a distinct measure which has to be selected with great care, because if the wrong measure has been optimized, the interactive correction of the result may require even more effort than the interactive vectorization of the original drawing.

3. OVERVIEW OF AUTOMATIC VECTORIZATION APPROACHES

In this section, published approaches to automatic vectorization will be described. Most methods will be analyzed to indicate why they are not fully suitable for solving the problem of vectorizing line drawings (specifically maps).

It should be noted that all but one algorithm in the following paragraphs describe vectorization approaches for line drawings with lines of one pixel thickness. Such lines can be acquired interactively, e.g., by using a drawing table or digitizing table (but then no automatic vectorization is necessary). The usual procedure to acquire line drawings is to scan them. In this case, scanned lines should have a thickness of more than one pixel, because otherwise a criterion similar to the Nyquist criterion will not be satisfied: to prevent aliasing, one has to scan with a resolution of at least twice the minimum line thickness.

To apply the algorithms to this type of drawing, the lines in the scanned image have to be reduced to one pixel thick lines. This is usually done with a skeletonization algorithm. However, this has several disadvantages: corners and T-junctions are often distorted because they may consist of very short spurious or unnecessary lines, and skeletonization is sensitive to rotation. These small lines can often be removed with specially designed rules, see for example Fig. 1. The algorithm described in this paper uses another approach to solve this problem.

Ideally, the most appropriate algorithm should be compared to a (commercial) system. However, this was not possible because we do not have such a system available in our laboratory. In [6] a commercial system for vectorizing line drawings is presented. An advantage of this system is that it has several subsystems for text and symbol recognition. Unfortunately, their vectorization method and clean up editing subsystem are not described. Also, no evaluation of the resulting vectorization is present. Thus, no comparisons could be made.

Sparse-Pixel Methods

Dori *et al.* [7] describe a method to vectorize engineering drawings, which does not require one pixel thick lines. This is done by scanning every so many row and column, thereby avoiding the necessity to address every pixel in the image. After a (line-like) blob of object pixels has been found, a ''zig-zag'' algorithm is used which follows the object pixels of the blob row-wise until a nonobject pixel is encountered, then it follows the pixels of the blob column-wise until a nonobject pixel is encountered, then row-wise, etc. until the end of the blob has been reached. Special provisions have been made for scanning horizontal and vertical lines. To detect junctions in the drawing, the average length of each row and column run is inspected. If a deviation is detected, a junction or change in line-thickness is assumed, and appropriate action is taken.

An advantage of this method is that most arc segments are detected as arc segments (and not as sequences of straight line segments). Also, the authors extend the method to recognize arrows in engineering drawings. However, a disadvantage of this method is that for the vectoriza-

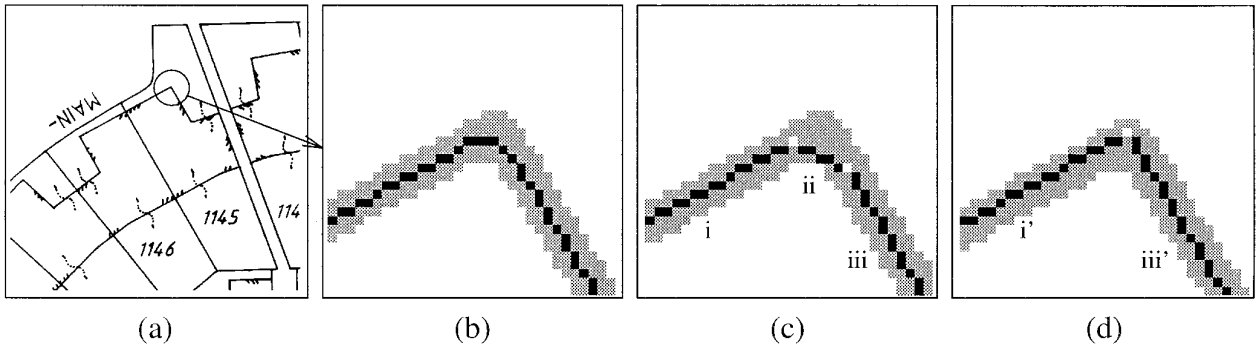


FIG. 1. An example illustrating how small edges in the vectorization can be removed: (a) a small part of a map, (b) enlargement of the skeletonization of (a) plotted in black on it, (c) the vectorization from the skeleton. Nodes are white, edges are black. Edge (*ii*) is unnecessary because it is short. By setting constraints on the angles of the edges involved and on the length of edge (*ii*), the vectorization can be corrected as shown in (d).

tion algorithm, approximately 12 thresholds must be adjusted in advance, and for the arrow recognition algorithm approximately 13 thresholds must be adjusted. It is not specified for most parameters how they should be set. Also, it is difficult to judge the performance since no quantitative measurement is present.

The remainder of this section describes vectorization approaches assuming lines in the input image are of one pixel thickness.

Area Methods

The Wall–Danielsson method [8] is an example of this type of methods. An extension is described in [9]. The criterion used is the maximum allowed area deviation per unit line length from the vectorization to the approximated curve. The last point on the curve for which this criterion is not violated becomes a node in the vectorization and this process is repeated until the end of the curve.

A principal difficulty is that nodes sometimes do not correspond with corners in the curve because a new edge is only started as soon as the criterion has been violated [10]. This may happen just a few pixels after a long straight line turns a corner, and not at the corner itself.

Cone Intersection Methods

These methods, presented by Williams [11, 12], and Sklansky and Gonzalez [13], are based on defining a circle around each point of the curve in the drawing. The approximating edge has to intersect all these circles. The name of this method originates from the form of the union of all these circles seen from the start point, which is a cone. In [12] the author improves his results obtained in [11] by allowing edges to end outside the circles. Thus, a property of this method is that nodes of the approximating edges are not necessarily part of the original curve. This is undesirable from the viewpoint of vectorizing maps.

Minimax Methods

These methods, presented by Kurozumi and Davis [14], are based on the minimization of the maximum distance between the original curve and the approximating edges. Consequently, nodes of the approximating edges are not necessarily part of the original curve as in the previous method.

Tolerance Strip Methods

Given a number of points and a maximum and minimum strip width, these methods (Dettori [15], Leung and Yang [16]) try to fit a strip around the points to be approximated. Several approaches can be taken: the number of original image points in a strip or the length of the strip can be maximized, or the width of the strip can be minimized. Combinations are also possible. The resulting edge is the central axis of the strip.

In [15, 16] it is not clear how branch points are handled. The algorithms construct a strip from a given start point to the next point until the criterion is violated. When the start point is a branch point, ambiguous situations may arise, especially when the maximum strip width is large.

Run-Length Methods

After computing the run-length encoding of an image, a line adjacency graph is constructed (see e.g., Pavlidis [17, 18]). This graph is used to determine the edges of the vectorization. This method may not be suitable for vectorizing maps because a slightly slanted line with a jagged boundary may generate far more than one edge. Furthermore, the resulting vectorization may be orientation dependent.

Mark and Sweep Methods

Using the coordinates of the first point of a curve and the chain code of the contour as input, the method of

Sirjani and Cross [19] is based on marking all the points that may be an end point of a straight line. In the next stage the superfluous marked points are discarded, and the remaining points are the nodes of the vectorization. This method is not suitable for vectorizing maps because the outer contour is used: drawings which have lines fully enclosed by other lines will not be vectorized correctly.

Curvature Minima and Maxima Methods

Points of minimum and maximum curvature in a curve can be used for constructing a vectorization. The methods of Teh and Chin [20] and Ray and Ray [21] do not require an input parameter: points are selected on the basis of their region of support. The idea is that these ''dominant points'' provide important information for the recognition process of the human visual system. The position of the dominant points strongly depends on the method by which the region of support is determined. If it is determined incorrectly, dominant points may be discarded. In [20], three different methods are used, but it is not clear which one is preferred.

Maximum Perpendicular Distance Methods

The Douglas–Peucker method [22] is an example of these methods. The maximum allowed perpendicular distance in pixels t_{dp} from the vectorization to the curve to approximate is used as a criterion. The algorithm starts with two dominant points of the curve as nodes of the vectorization. When the maximum perpendicular distance from this vectorization to the curve is larger than t_{dp}, a new node is added to the vectorization at the place of the maximum deviation. This process is repeated recursively.

Other authors using this principle include Leu and Chen [23], who concentrate on uniqueness of the approximating vectorization and its accuracy, and Ramer [24].

Because of its strong tendency to put the nodes on the corners of the original image it appears that the Douglas–Peucker method is most suitable for vectorizing maps compared to the other algorithms. Therefore, it is chosen as the basis for our adaptive vectorization algorithm.

4. ADAPTIVE VECTORIZATION ALGORITHM

The outline of the adaptive vectorization algorithm is given in Fig. 2. It consists of three modules: generation of a coarse vectorization, construction of the anchor points from this vectorization, and finer vectorization between the anchor points.

Inputs and Outputs

The inputs to the algorithm are the scanned input image, the global line thickness w_{glob}, and a threshold for the refined vectorization algorithm t_{dp}. Outputs are two data structures containing the anchor points and the final vectorization.

Small Object Removal

To detect the small objects in the image, a histogram is made from the length of the diagonal of the bounding box of each connected component (the bounding box has axes parallel to the x- and y-axis). Small objects (characters, small blobs, etc.) are divided among the lower bins while the larger (lines, etc.) are divided among the upper bins. The bin of the first peak in the histogram is used to find the bin separating the small and large objects: from the bin of the peak a search is done for a series of consecutive empty bins. This number of consecutive empty bins is determined by the minimum of the bin number having the peak, and the value of the peak.

The last bin of the series of empty bins is used to separate the small objects from the large objects. Next, the small objects can be isolated or removed.

This module is not used in the adaptive vectorization algorithm, but it is used for the evaluation and to determine the global line thickness.

Determination of the Global Line Thickness

The global line thickness (w_{glob}) is determined as follows. First, the small objects are removed from the input image. The number of object pixels (N_{OP}) in the resulting image is counted. Then, after skeletonization, the image is vectorized with the Douglas–Peucker algorithm with $t_{\mathrm{dp}} = 1\frac{1}{2}$ pixels. Such a small threshold is used to obtain a precise vectorization. Next, the total Euclidean length of the vectorization T_{EL} is determined (that is: the added lengths of all vectors). The first estimation of the global line thickness is computed as $w_{\mathrm{glob-init}} = N_{\mathrm{OP}}/T_{\mathrm{EL}}$. Next, a correction δ is determined: $\delta = \frac{1}{2}w^2_{\mathrm{glob-init}}$. Also, the number of end points (N_{E}), T-junctions (N_{T}), and X-junctions (N_{X}) is determined. Each end point contributes δ to N_{OP}, and each T-junction and X-junction contributes $-\delta$ and -2δ, respectively. Eventually, w_{glob} (a floating point value) is computed as

$$w_{\mathrm{glob}} = \frac{N_{\mathrm{OP}} - N_{\mathrm{E}}\delta + N_{\mathrm{T}}\delta + 2N_{\mathrm{X}}\delta}{T_{\mathrm{EL}}}.$$

Generation of a Skeleton Image

An image with one pixel thick lines is produced with a skeletonization algorithm followed by removal of skeleton branches shorter than $2w_{\mathrm{glob}}$. The skeletonization is a pseudo-Euclidean skeleton [25]. This is an improvement over the Hilditch skeleton [26] because a better (almost Euclidean) metric is used. This ensures that the skeletonization is less sensitive to rotations.

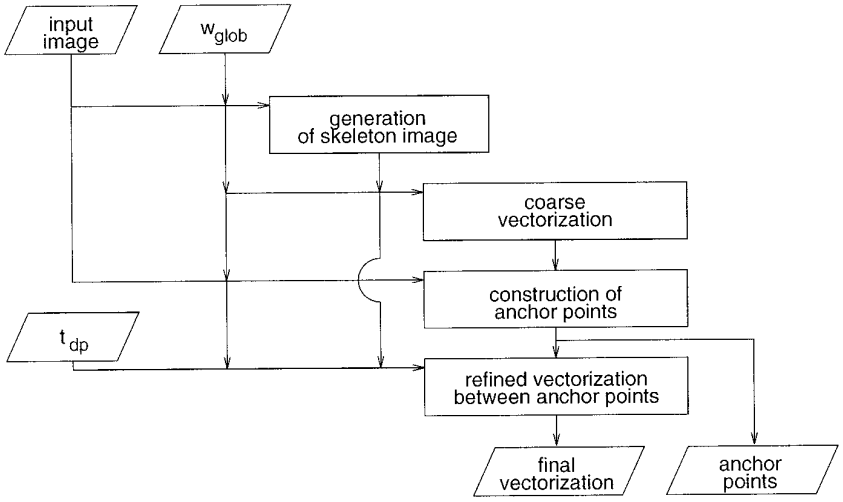


FIG. 2. A scheme of the adaptive vectorization algorithm.

Generation of a Coarse Vectorization

The skeleton image is vectorized with the Douglas–Peuker algorithm as implemented in the CARV package [27], with (large) $t_{dp-coarse} = \lfloor 2w_{glob} \rfloor$ pixels. The result is a coarse vectorization in which the *dominant points* (end points, corners, junctions) are captured. Because of the large threshold, small variations are not captured.

Construction of Anchor Points

The dominant points are moved to their ''correct'' location. The correct location is the middle of the intersection of the lines formed by the pixels in the bitmap image: the place where nodes ''should be.'' At this stage they are called *anchor points,* to indicate the idea that they give a coarse outline of the vectorization of the image, and that they do not move in the next module, which refines the vectorization.

Adjustment of the dominant points is done using the coarse vectorization. A region of interest is defined around each node in the original image in that vectorization (the node is assumed to be approximately at the correct position). Then, a structuring element is constructed from the node and the edges originating from it. This is matched with maximum threshold morphology on the region of interest. A better match is expected and this process is repeated until no improvements are observed. The construction of the anchor points is explained in detail in Section 4.1.

Refined Vectorization between the Anchor Points

This is the last module in the vectorization algorithm. Its inputs are the vectorization with anchor points, the skeleton image, w_{glob}, and the threshold for the vectorization algorithm t_{dp}. The purpose of this module is to improve the accuracy by reducing the deviation from the lines in the skeleton image.

To this end, the skeleton image is again vectorized, but now with threshold t_{dp}. For each node, the distance to the closest anchor point is determined. If this distance is $\leq$ w_{glob} pixels, the node is merged with the closest anchor point. If t_{dp} is small enough ($1\frac{1}{2}$ pixels is often a good value), fine details in the image will be captured. This is different from the module to generate the coarse vectorization, in which a large $t_{dp-coarse}$ was used to capture the gross outline. At this stage the details are important.

Our approach is motivated by the idea that the vectorization should represent the lines in the original image. The vectorization may have been distorted by the algorithm used. By comparing it with the original image, inconsistencies can be detected. It seems logical to do this only for the nodes of the coarse vectorization, because these capture the gross outline of the lines in the original image. They also provide important information for the recognition process of the human visual system [28]. Another advantage is that it is computationally more efficient to do this only for these nodes. In the next phase, the details can be captured. In our approach, this is done by the refined vectorization between the anchor points.

4.1. *Construction of Anchor Points*

In the second module in the vectorization algorithm, the inputs are the coarse vectorization, the original input image, and the global line thickness w_{glob}. Nodes in the coarse vectorization are moved to their ''correct'' location—the intersection of the lines in the original image—using maximum threshold morphology. After that they are called anchor points.

Maximum threshold morphology is a modified version of threshold morphology [29]. A small image, called a *structuring element,* is constructed with odd-sized width and height and integer-valued pixels. With threshold morphology, a threshold t_{tm} is set in advance. Each pixel in the binary input image is convolved with the structuring element. If this results in a pixel value larger than or equal to t_{tm}, the pixel in the binary output image becomes 1, otherwise it becomes 0. With *maximum* threshold morphology, each pixel in the binary input image is convolved with the structuring element, resulting in an output image with integer valued pixels. Then, the maximum value of that output image is determined, and the output image is thresholded with that value so that a binary image remains, in which the object pixels are the pixels of maximum value in the integer valued output image.

Determination of the size of the structuring element and the search region. The thickness of the lines in the original image should have a direct influence on the construction and the size of the structuring element, because otherwise the maximum threshold morphology may give wrong results. To improve accuracy, the local line thickness w_{loc} of the lines in the original image is determined for each node in the coarse vectorization. This is done using the original image and a pseudo-Euclidean skeleton in a region of interest around the node: $w_{\text{loc}} = N_{\text{OP}}/N_{\text{SP}}$, with N_{OP} the number of object pixels, and N_{SP} the number of skeleton pixels in the region of interest. w_{loc} is a floating point value, just as w_{glob}. This approximation method is used instead of the more precise approximation method for w_{glob}, because it can be computed fast. Also, it is better to use a value which is approximately correct than to use a value (w_{glob}) which may be incorrect. For instance, in Fig. 7d, the local line thickness at the large blobs is much larger than w_{glob}. Eventually, w_{loc} is used to determine the size of the structuring element $\text{size}_{\text{strelm}}$:

$$\text{size}_{\text{strelm}} = \begin{cases} \lfloor 7w_{\text{loc}} \rfloor & \text{if } \lfloor 7w_{\text{loc}} \rfloor \text{ odd} \\ \lfloor 7w_{\text{loc}} \rfloor + 1 & \text{if } \lfloor 7w_{\text{loc}} \rfloor \text{ even} \end{cases}$$

The size of the search region is determined as $2 \times \text{size}_{\text{strelm}} - 1$. This is the same size as the structuring element, enlarged to allow a border to prevent boundary effects.

Construction of the structuring element. The construction of the structuring element is illustrated with an example in Fig. 3. In (a), the candidate anchor point and its edges in the region of interest are drawn in a binary image. This image is dilated for a total of $\lfloor (w_{\text{loc}} - 1)/2 \rfloor$ iterations, shown in (b), using 8- and 4-connectivity alternately, to largely suppress the effects of grid anisotropy [30]. This number of iterations was taken because the width of the object in the dilated image is just a bit smaller than the local line thickness of the lines at that point in the original image. The object pixels of (b) are copied into the structuring element, and assigned the value 1, as shown in (e). Next, the outer contour of (b) is determined, as shown in (c). These pixels are assigned the value 0 in the structuring element (white pixels—invisible—in (f)). Finally, a border of thickness $\lfloor w_{\text{loc}} \rfloor$ is constructed from the contour (again using dilation) shown in (d). This border is copied to the structuring element and its pixels are assigned value -1. This is shown in gray in (g). Positive valued pixels indicate a positive, negative a negative, and zero valued a don't care match.

Three-valued structuring element. A three-valued structuring element was used because a binary-valued did not suffice. This is illustrated in Fig. 4. In (a) the binary input image (and the search region) for the maximum threshold morphology is given. Both a corner and a T-junction are in the search region, and they are very similar except for the third arm of the T-junction. To find the anchor point for the corner, a structuring element is constructed from the node in the coarse vectorization at that corner and its edges.

This is done for example in (b), by dilating the edges from the coarse vectorization. The result of the maximum threshold morphology on (a) is given in a 3D plot in (c) (the values in the 3D plot represent the integer pixel values of the output image, just before the thresholding operation). There are two distinct, but equal-valued maxima, one for the corner and one for the T-junction. By using a three-valued structuring element, this can be prevented: in (d) a structuring element is shown, constructed according to the method described above. The result of the maximum threshold morphology on (a) is shown in (e). Now, there is only one maximum (the left peak). In conclusion, a three-valued structuring element prevents similar but slightly different structures in the search region from generating more than one maximum.

Determination of the anchor points. The determination of the anchor points is illustrated with an example in Fig. 5. In (b) an enlargement of (a) is given. The following is a description of what happens in the dotted circle in (b). In (c) the result of the coarse vectorization is given: the original image is shown in gray, the edges of the vectorization are shown in black, and the node of the vectorization is shown in white.

A structuring element is constructed from node i and its coarsely vectorized edges. The result is depicted in (d). Pixels in black have a positive value (1), pixels in gray a negative value (-1), and the other pixels a zero value. The new point resulting from the maximum threshold morphology with (d) on (b) is given in (e) (point *ii*). Its edges are

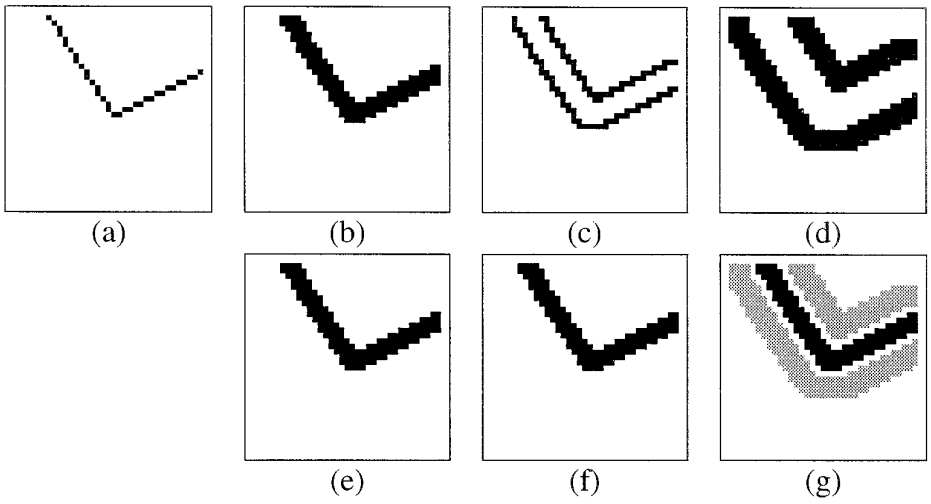


FIG. 3. An example construction of the structuring element. The top row depicts a temporary binary image, and the bottom row depicts the structuring element itself. Pixels in black have value 1, in white value 0, and in gray value −1. For explanation see the text.

also drawn to emphasize that the point is only moved. Change is observed so the procedure is repeated.

A new structuring element is constructed from the edges and the new node *ii* in (e), shown in (f) (note that (f) is slightly different from (d) because node *i* has moved to *ii*). After the maximum threshold morphology, the new point is *iii* in (g). Again, change is observed so the procedure is repeated.

The new structuring element generated from node *iii* and its edges is shown in (h). Again, (h) is slightly different from (f) and (d). After the maximum threshold morphology, the new node found is the same as the node in the previous iteration (*iii* and *iv* are the same), so no change is observed and the process stops. Finally, node *iv* is called the anchor point. In (j) the generated points can be compared with each other.

The process not only stops when the points from two subsequent matches are the same, but also when they are 4-connected to each other, or after a fixed number of iterations to prevent repetitive looping. The latter may happen

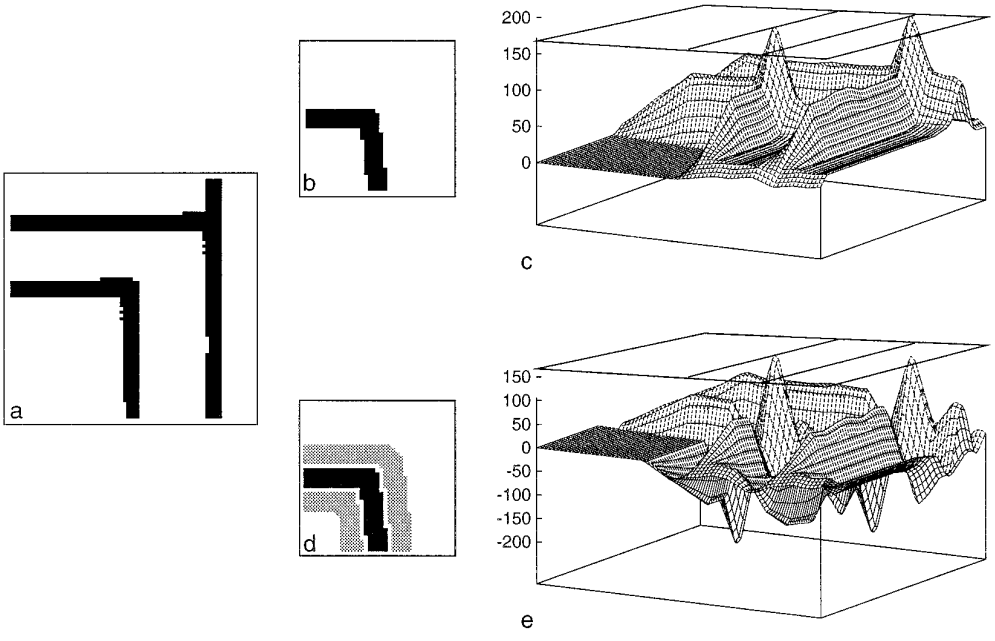


FIG. 4. A binary valued structuring element does not suffice for the maximum threshold morphology. See the text for details.

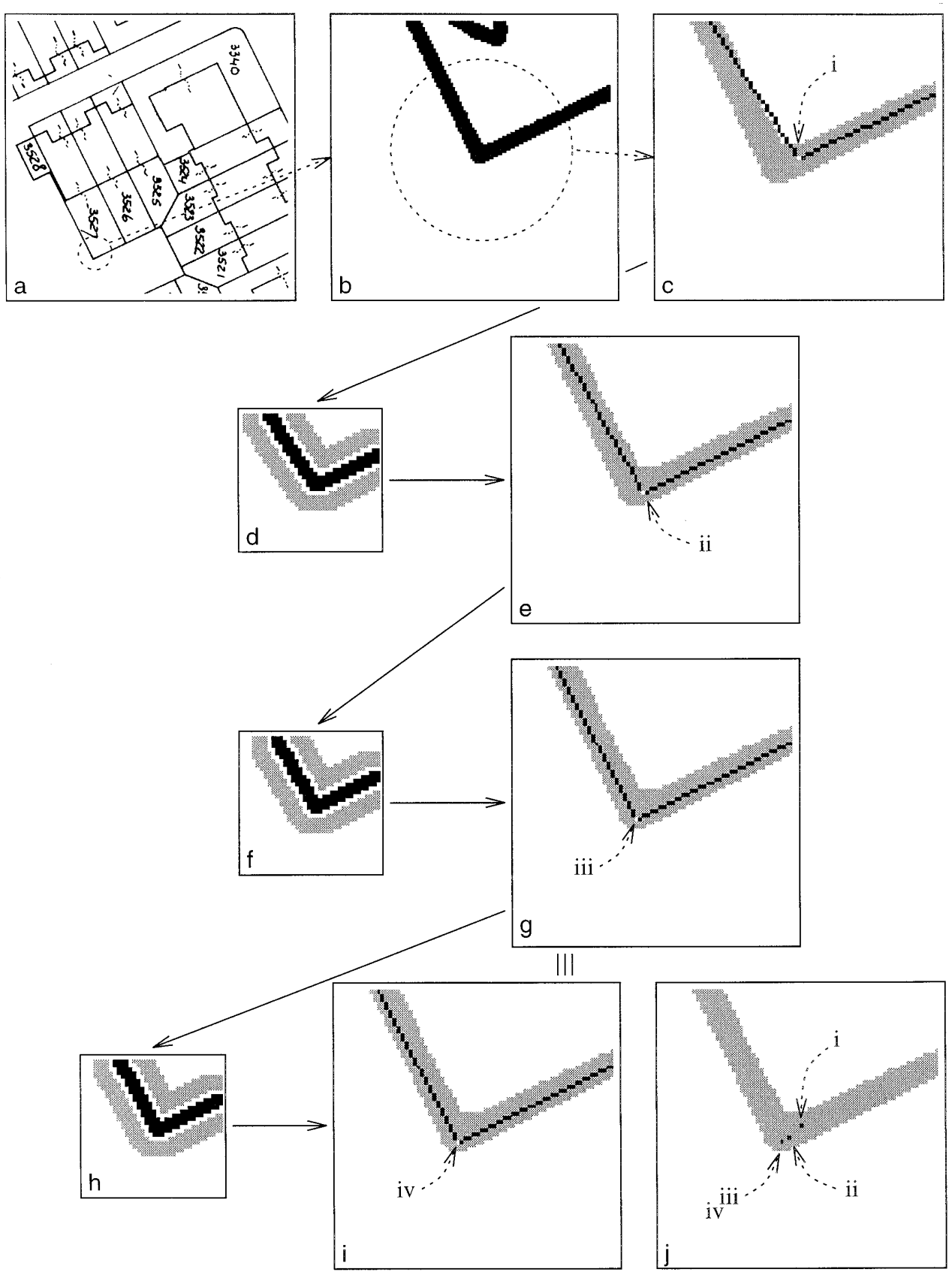


FIG. 5. An example illustrating how anchor points are constructed. See the text for explanation.

if e.g., a node at location A is moved to location B, in the next match the same node is moved from location B to C, then from C to D, and finally from D back to A.

5. EXTENDED ADAPTIVE VECTORIZATION ALGORITHM

Sometimes, additional postprocessing is desired. For instance, on the map in Fig. 1a, there are very short parallel lines extending from long lines. These are hatches used on some maps to indicate the inside of the lines enclosing buildings. We call them *building hatches.* Because they do not contribute to the topology of the objects, one may prefer to remove them. The algorithm presented in the preceding section offers different possibilities for this kind of processing. Depending on the application, postprocessing modules can be added after each standard module, as follows.

Postprocessing of the Coarse Vectorization

This may be useful for example with X-crossings. Often these are not vectorized as an X-crossing but as two separate T-junctions connected by a very short edge. Although the construction of anchor points module will probably merge them to one node, computation time is saved if they are merged in advance, as merging nodes is computationally less expensive than maximum threshold morphology.

Postprocessing of the Anchor Points

Link nodes can be removed if the bend angle α between the two edges is less than a certain threshold, e.g., $|\alpha| \leq 15°$. This can be done when that anchor node does not capture the gross outline of the image.

The problem of removing the building hatches as described above can be solved by removal of very short edges in a module to postprocess the refined vectorization, followed by replacing them with one longer edge. Anchor points should not be removed: this can be implemented using information in the anchor points structure.

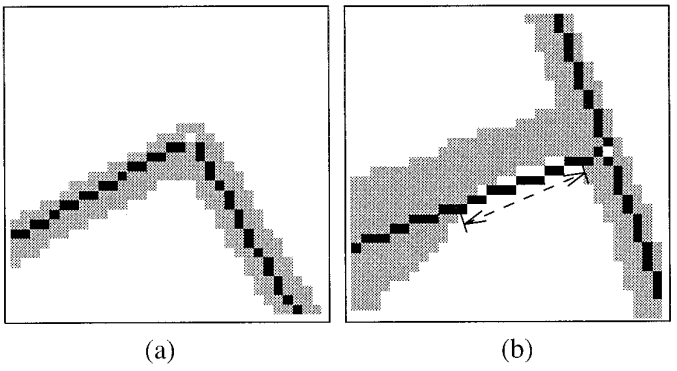


FIG. 6. (a) Example of a correct vectorization, (b) example of a wrong vectorization (next to the dashed arrow). Nodes of the vectorization are white, edges are black. The original bitmap is given in gray.

6. EVALUATION

6.1. *Error Measurement Method Used*

For the evaluation of the vectorization, we use the evaluation strategy from [3], which defines a *vectorization evaluation criterion* as follows:

The vectorization is considered satisfactory if, when it is plotted in the original image, the original image overlaps the plotted vectorization.

An example of this is shown in Fig. 6. The error is measured in two ways: the number of wrong pixels and the number of wrong lines. The number of nodes and edges in the vectorization is also reported.

Number of wrong pixels. A ''wrong pixel'' is defined as a pixel of the vectorization which is not covered by the original bitmap. The percentage wrong is calculated with $N_W/N_T \times 100\%$, with N_W the total number of wrong pixels, and N_T the total number of pixels in the vectorization. In Fig. 6b, 11 pixels are wrong on a total of 56.

Number of wrong lines. A ''wrong line'' is defined as a line of maximum length which can be formed from connecting ''wrong pixels.'' Such a line is always part of an

TABLE 1
General Information about the Line Drawings Used for Evaluation

Map	Figure	Drawing method	No. of pixels				Resolution (dpi)	Scale	w_{glob} (pixels)
			x size	y size	total	object			
geo	7a	template	1970	2400	4,728,000	364,198	508	unknown[a]	5.4
Leiden	7b	hand	2800	2700	7,560,000	571,297	400	1:1000	4.3
gbkn	7c	machine	3400	5600	19,040,000	780,983	400	1:500	4.5
scheme	7d	machine	4011	5192	26,017,112	993,442	400	—	6.3

[a] The scale is not written on the map. Unfortunately, its location is also unknown, thus it is not possible to determine the scale by measurements in the real world and compare this with measurements on the map.

edge of the vectorization. Reported is the number of wrong lines of each length, and their length. In Fig. 6b, there is one line wrong with a length of 11 pixels.

6.2. *Description of the Line Drawings Used*

Different line drawings (mostly maps) on different scales and resolutions were used for evaluation. This is summarized in Table 1. The maps ''geo'' and ''Leiden'' are cadastral maps, map ''gbkn'' is a large scale base map of the Netherlands (abbreviated in Dutch as G.B.K.N.), and ''scheme'' is an electrical diagram (see Fig. 7). The scanners used were for map geo a Scitex ELP 2 scanner, for map Leiden a Vidar A0 scanner, and for map gbkn and schematic diagram scheme a Hewlett–Packard Scanjet IIc. The entry ''template'' in the column ''drawing method'' indicates that the map was drawn by hand, but that the characters and numbers were written using a template.

6.3. *Experiments*

For evaluation, the four different line drawings were vectorized with the algorithms described in the previous sections. These are compared with results in [3]. All images were preprocessed by automatic removal of small objects (e.g., characters, numbers). This was done because the vectorization algorithm must be evaluated, and vectorization of small objects does not give an accurate measurement of the performance of the vectorization algorithm.

The experiments are labeled with the name of the line drawing followed by an extension, printed in italics. The extensions are:

• extension *dp*. After preprocessing, the image is vectorized using the standard Douglas–Peucker algorithm with $t_{\text{dp}} = 1\frac{1}{2}$ pixels.

• extension *adap.* After preprocessing, the image is vectorized with the adaptive vectorization algorithm described in Section 4. We used $t_{\text{dp}} = 1\frac{1}{2}$ pixels in the module to refine the vectorization.

• extension *ext.* After preprocessing, the image is vectorized with the adaptive vectorization algorithm described in Section 4 and the postprocessing modules described in Section 5. All experiments with extension *ext* have the same postprocessing modules for the coarse vectorization and for the anchor points.

Postprocessing of the coarse vectorization. Postprocessing of the coarse vectorization merges two T-junctions connected with an edge of length $\leq w_{\text{glob}}$ into one X-junction.

Postprocessing of the anchor points. Postprocessing of the anchor points consists of two operations: two T-junctions connected with an edge $\leq w_{\text{glob}}$ are merged to one X-junction, and link nodes are removed if the bend angle α between the two edges is $|\alpha| \leq 15°$.

Postprocessing of the refined vectorization. Postprocessing of the refined vectorization differs for each of the four line drawings as follows. In all of these experiments, the refined vectorization was computed with $t_{\text{dp}} = 1\frac{1}{2}$ pixels.

—experiment *geo–ext.* Because map geo has building hatches, it is expected that too many nodes and edges will be generated. To handle this, the postprocessing module removes edges connected to link nodes with length $\leq 2w_{\text{glob}}$, until a nonlink or anchor node is encountered. The two extremes are then reconnected with one (longer) edge. Nonanchor link nodes also are removed if the bend angle α between the two edges is $|\alpha| \leq 5°$.

—experiment *leiden–ext-one.* No postprocessing of the refined vectorization is done.

—experiment *leiden–ext-two.* Nonanchor link nodes are removed if the bend angle α between the two edges is $|\alpha| \leq 5°$.

—experiment *gbkn–ext.* No postprocessing of the refined vectorization is done.

—experiment *scheme–ext.* No postprocessing of the refined vectorization is done.

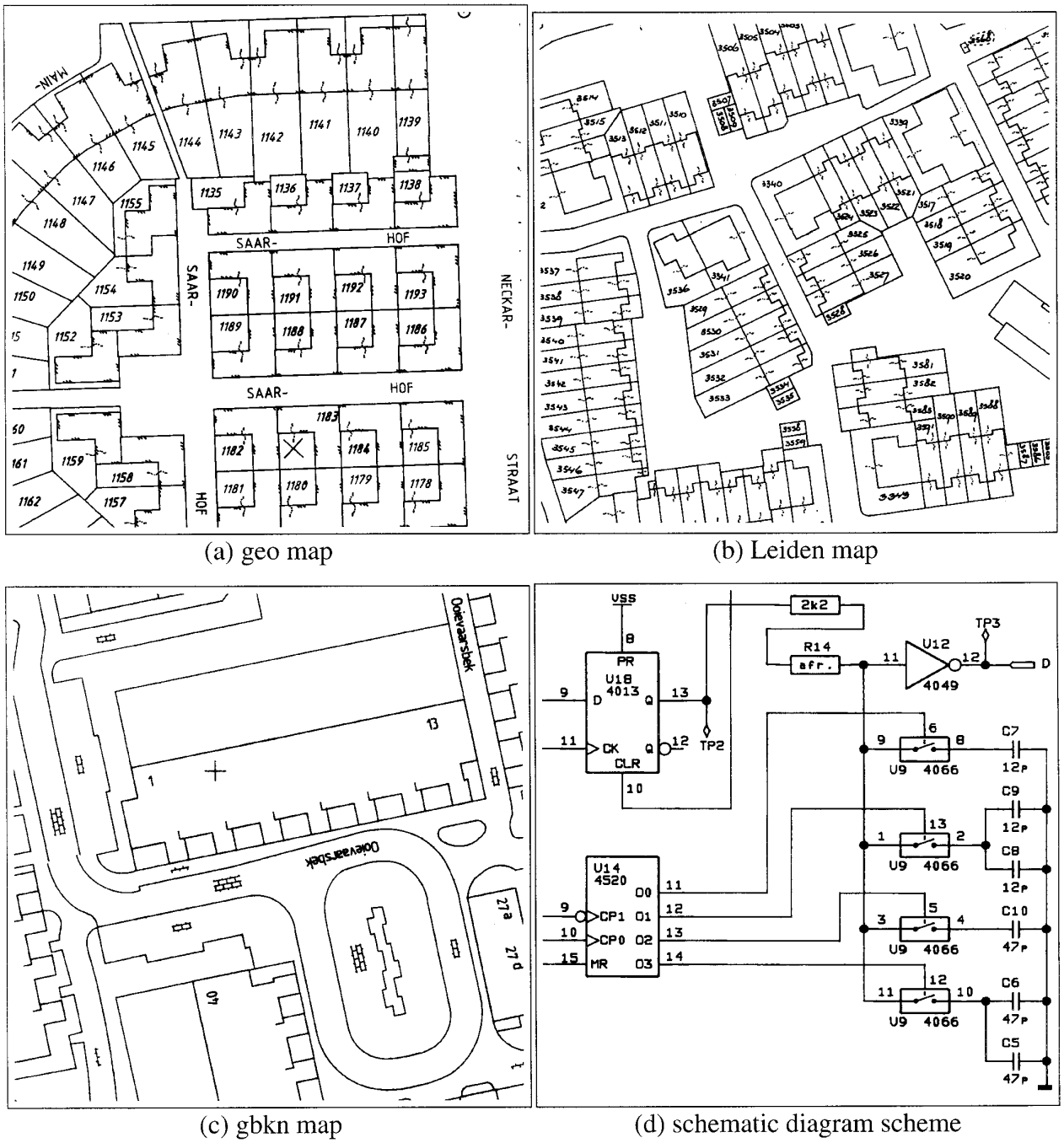
• extension *clean.* After preprocessing, the image is vectorized with the adaptive vectorization algorithm described in Section 4 and the postprocessing modules described in Section 5.

Postprocessing of the coarse vectorization. The postprocessing of the coarse vectorization consists of removing link nodes if the bend angle α between the two edges is $|\alpha| \leq 15°$, and merging the endpoints of an edge of length $\leq \lfloor 2w_{\text{glob}} \rfloor$ to a new node in the middle of that edge (and effectively removing that edge). Using this last rule, two T-junctions are merged to one X-junction.

Postprocessing of the anchor points. No postprocessing of the anchor points is done.

Postprocessing of the refined vectorization. The postprocessing of the refined vectorization consists of two parts. To compute the refined vectorization, we used $t_{\text{dp}} = 1\frac{1}{2}$ pixels. The first part removes edges connected to nonanchor link nodes and reconnects the endpoints of these two edges with a longer edge. This is only done if the error according to the vectorization evaluation criterion is 0 for all edges involved. So the error cannot change using this clean up rule, only the number of nonanchor nodes can change. The second part moves a node in the eight chain code directions under the condition that the error of the attached edges decreases. At the location of the coordinates of the new node must be an object pixel in the original bitmap. This is done first for nonanchor nodes, and then for anchor nodes.

• extension *icdar93.* These are the results of the vectori-



(a) geo map (b) Leiden map

(c) gbkn map (d) schematic diagram scheme

FIG. 7. Sections of the line drawings used for evaluation.

zation described in [3]. In that paper, the vectorization was refined using special rules, see e.g., Fig. 1. Only the results for the "geo" and the "Leiden" maps can be described because only those maps have been evaluated. In that paper, edges from the vectorization connected to end points are removed. Thus, only polygons remain in the vectorization. This was done to prevent border effects. This explains the difference in the number of pixels of the vectorization. In contrast, in this paper the full vectorization is evaluated.

6.4. *Results and Discussion*

The total number of pixels in the vectorization (N_T), the number of wrong pixels in the vectorization (N_W), the number of nodes and edges, and the number of wrong

TABLE 2
The Number of Wrong Pixels in the Vectorization, the Number of Nodes and Edges, and the Number of Wrong Lines Sorted on Their Length in Pixels

	No. of pixels					No. of lines of length				
Map	N_T	N_W	%	No. of nodes	No. of edges	≥1	≥2	≥3	≥5	≥10
geo–dp	49,181	17	0.0	1845	1949	17	1	1	0	0
geo–icdar93	45,351	368	0.8	1412	1499	55	32	20	12	5
geo–adap	49,552	445	0.9	1613	1717	49	27	19	11	7
geo–ext	49,315	353	0.7	599	685	45	32	16	11	5
geo–clean	49,520	15	0.0	481	586	2	1	1	1	1
leiden–dp	100,131	89	0.1	2537	2814	30	11	7	5	0
leiden–icdar93	90,574	1119	1.2	1653	1945	315	156	85	49	17
leiden–adap	101,032	620	0.6	2178	2454	82	44	30	17	8
leiden–ext-one	101,042	431	0.4	2183	2459	85	45	28	16	5
leiden–ext-two	101,042	825	0.8	1860	2136	88	50	33	19	9
leiden–clean	100,959	51	0.1	1437	1714	14	4	2	1	0
gbkn–dp	138,379	12	0.0	2384	2469	6	1	1	0	0
gbkn–adap	139,263	154	0.1	2222	2307	55	34	19	5	0
gbkn–ext	139,268	120	0.1	2222	2307	54	32	18	4	0
gbkn–clean	139,226	0	0	1362	1447	0	0	0	0	0
scheme–dp	104,712	158	0.2	982	1059	2	2	2	2	2
scheme–adap	105,381	213	0.2	754	831	10	9	8	4	3
scheme–ext	105,392	213	0.2	736	831	10	9	8	4	3
scheme–clean	105,369	2	0.0	558	635	1	1	0	0	0

lines sorted by their length in pixels are shown in Table 2. The error is smallest with the *clean* experiment, followed by the standard Douglas–Peucker algorithm. The generated number of nodes and edges is for all our experiments less than with the standard Douglas–Peucker algorithm.

The (a priori) expectation of the result from the vectorization of the ''geo'' map (a large number of nodes and edges caused by the building hatches) was correct: the performance of experiment *geo–ext* (which used rules to correct for building hatches) was better than experiment *geo–adap*. Also, the number of nodes and edges decreased.

By comparing experiments *leiden–ext-one* and *leiden–ext-two* it can be observed that removing nonanchor link nodes, if the bend angle α between the two edges is $|\alpha| \leq 5°$, is risky, but useful: the number of nodes and edges decreases while the error increases slightly for the number of wrong lines (the error for the number of wrong pixels almost doubles).

There is no difference between experiments *gbkn–adap* and *gbkn–ext,* and between experiments *scheme–adap* and *scheme–ext.* Thus, the proposed postprocessing modules in experiments *gbkn–ext* and *scheme–ext* have almost no effect. To improve the results, a different approach must be taken, for example as in experiments *clean.*

The best results are obtained with the *clean* experiments. These experiments have the advantage that the application of the clean-up rules for the refined vectorization guarantee that the error only decreases. In Figs. 8–11 for each of the four line drawings a very small part (500 × 350 pixels) of the result of the vectorization with the Douglas–Peucker algorithm and with the *clean* experiment can be compared. The original bitmap is given in gray, and the vectorization in black. Nodes are indicated by small + signs.

The execution time of the algorithm for the *dp* and *clean* experiments is given in Table 3. These times were measured on a SUN Sparc 20, with 256 Mb memory. Experiment *scheme–clean* takes more time to complete than experiment *gbkn–clean,* partly because the scheme drawing is larger, but also beause w_{glob} for the first experiment is larger than for the latter. This results in a larger size of the structuring element and of the search region, and this will result in more time necessary to compute the maximum threshold morphology.

In designing the algorithm, we emphasized accuracy rather than execution time. Therefore, the algorithm is not very fast. This is caused by the two applications of the Douglas–Peucker algorithm, the number of morphological operations (for each anchor point a few have to be computed), and the time necessary for the clean-up rules. However, interactive vectorization is most often slower and (thus) more expensive. To obtain the same speed as the computer, humans should label about 2 to 3 nodes per

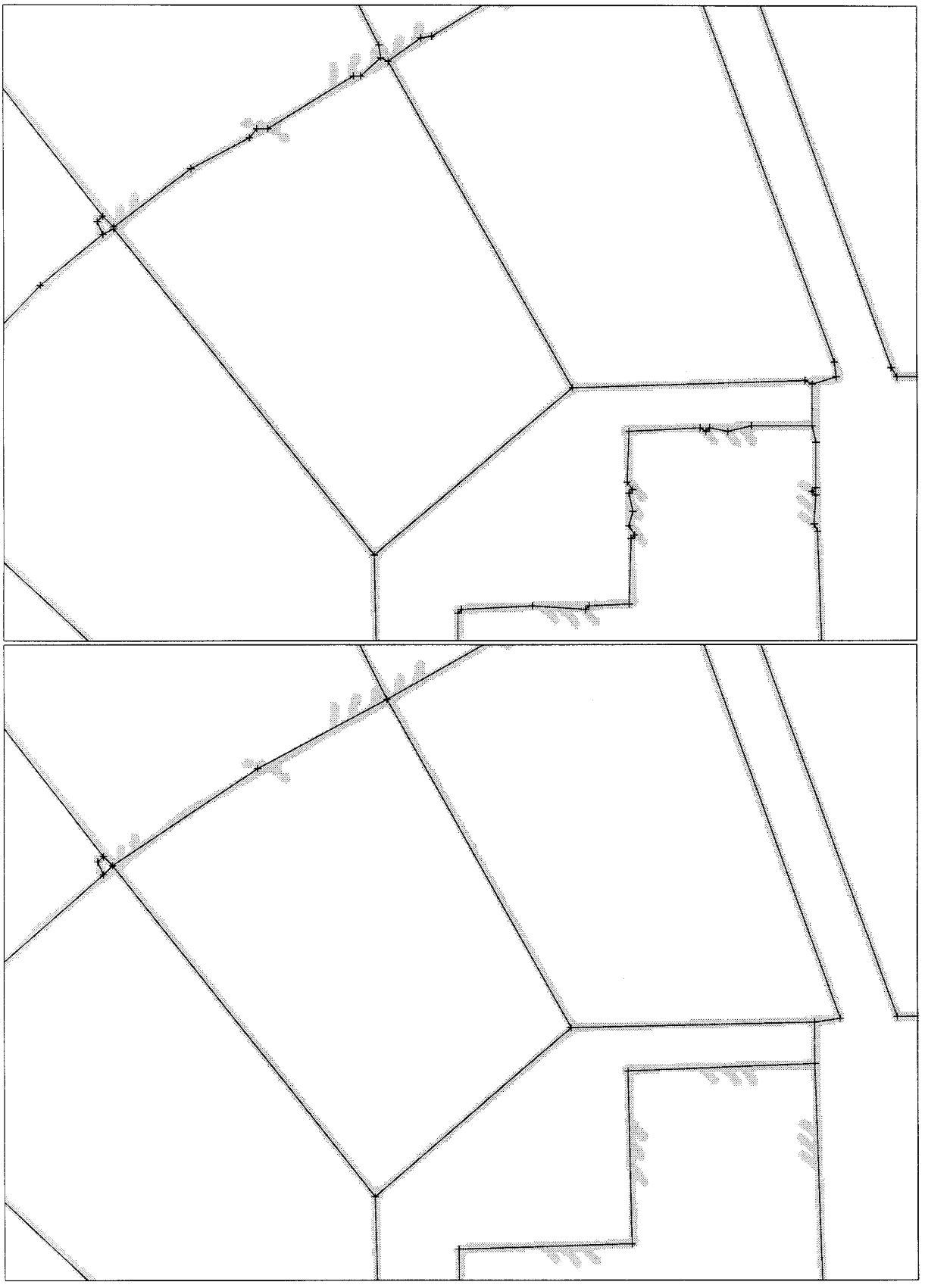


FIG. 8. Fraction of the results of experiment *geo–dp* (top) and *geo–clean* (bottom).

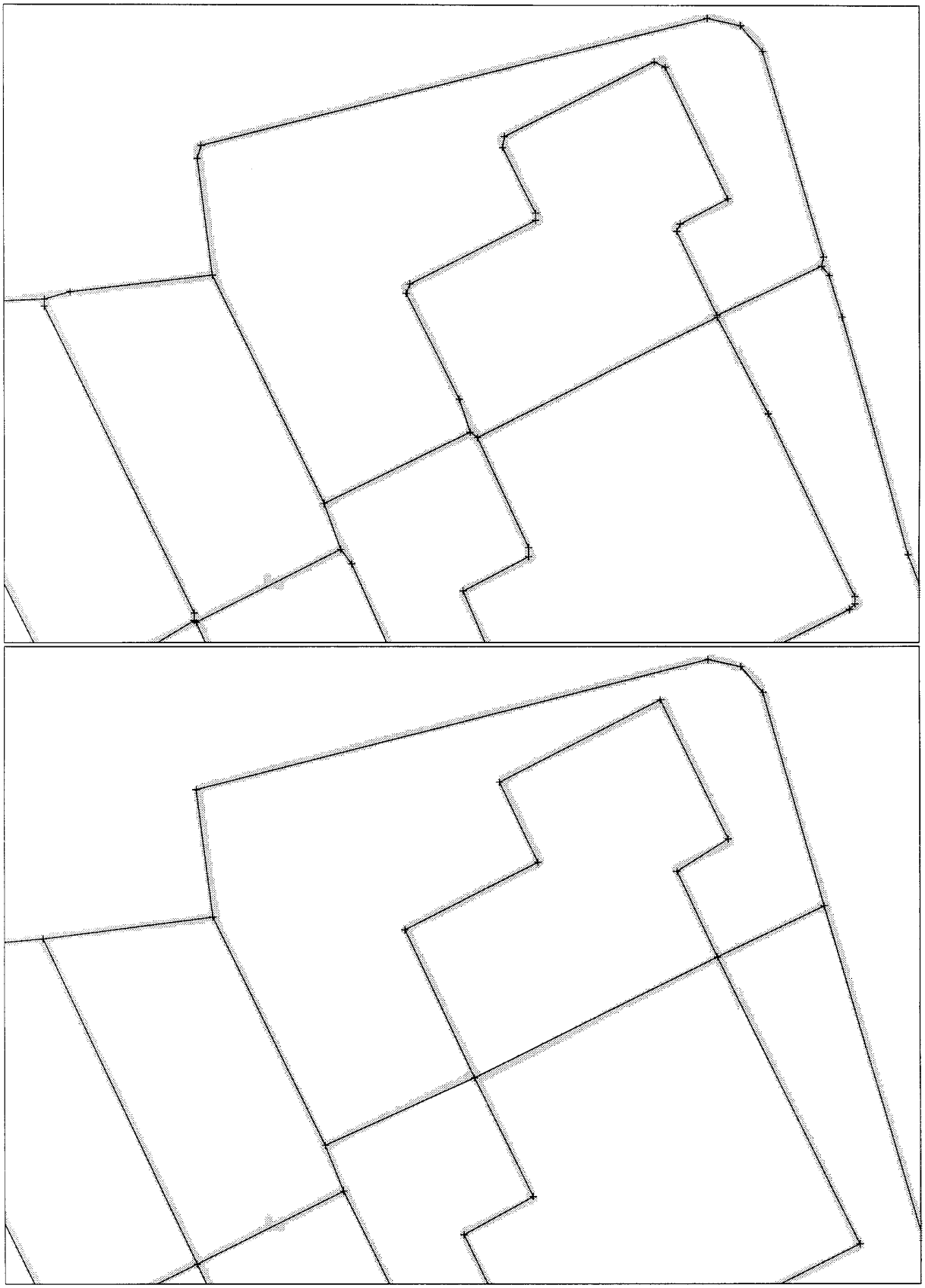


FIG. 9. Fraction of the results of experiment *leiden–dp* (top) and *leiden–clean* (bottom).

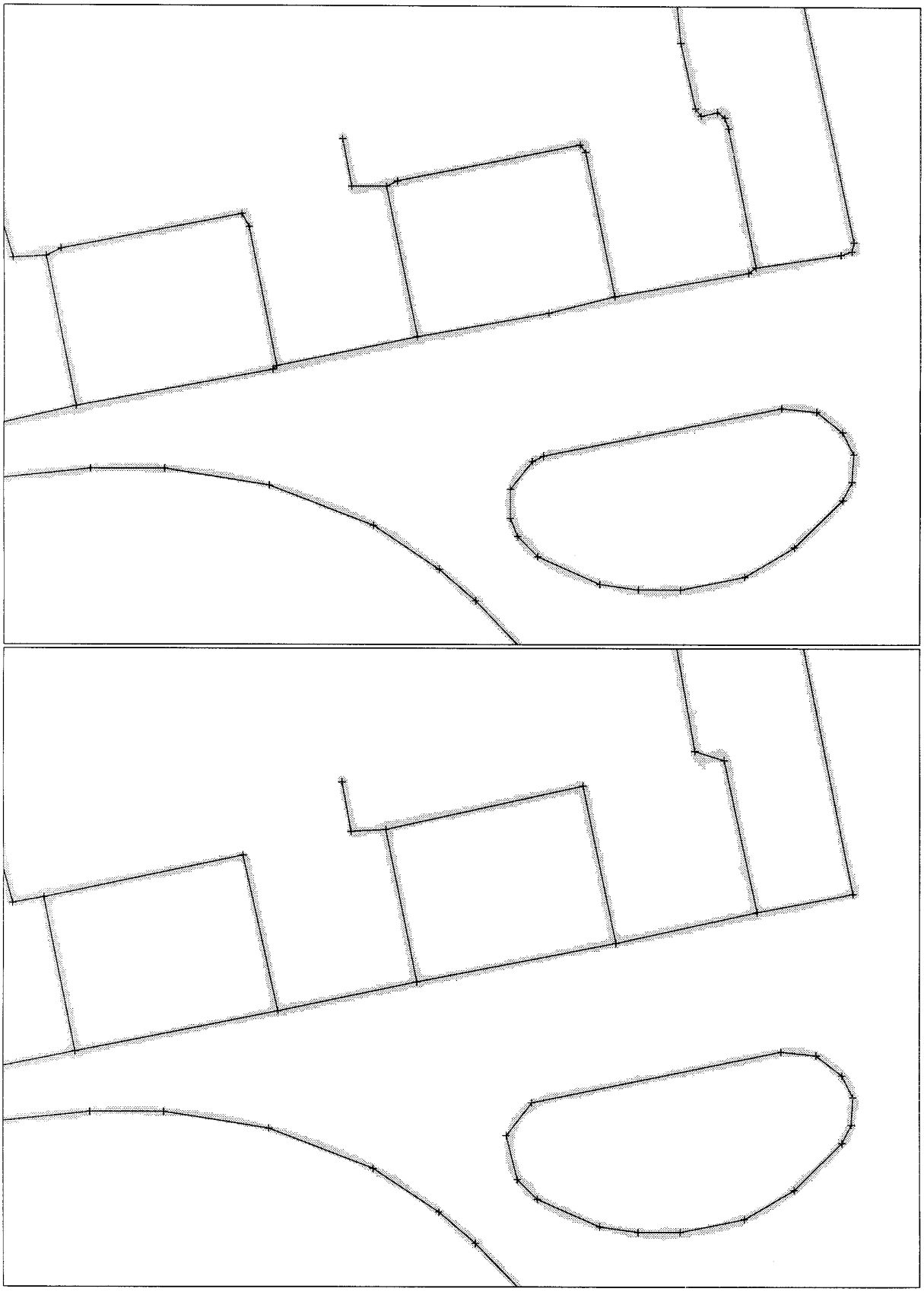


FIG. 10. Fraction of the results of experiment *gbkn–dp* (top) and *gbkn–clean* (bottom).

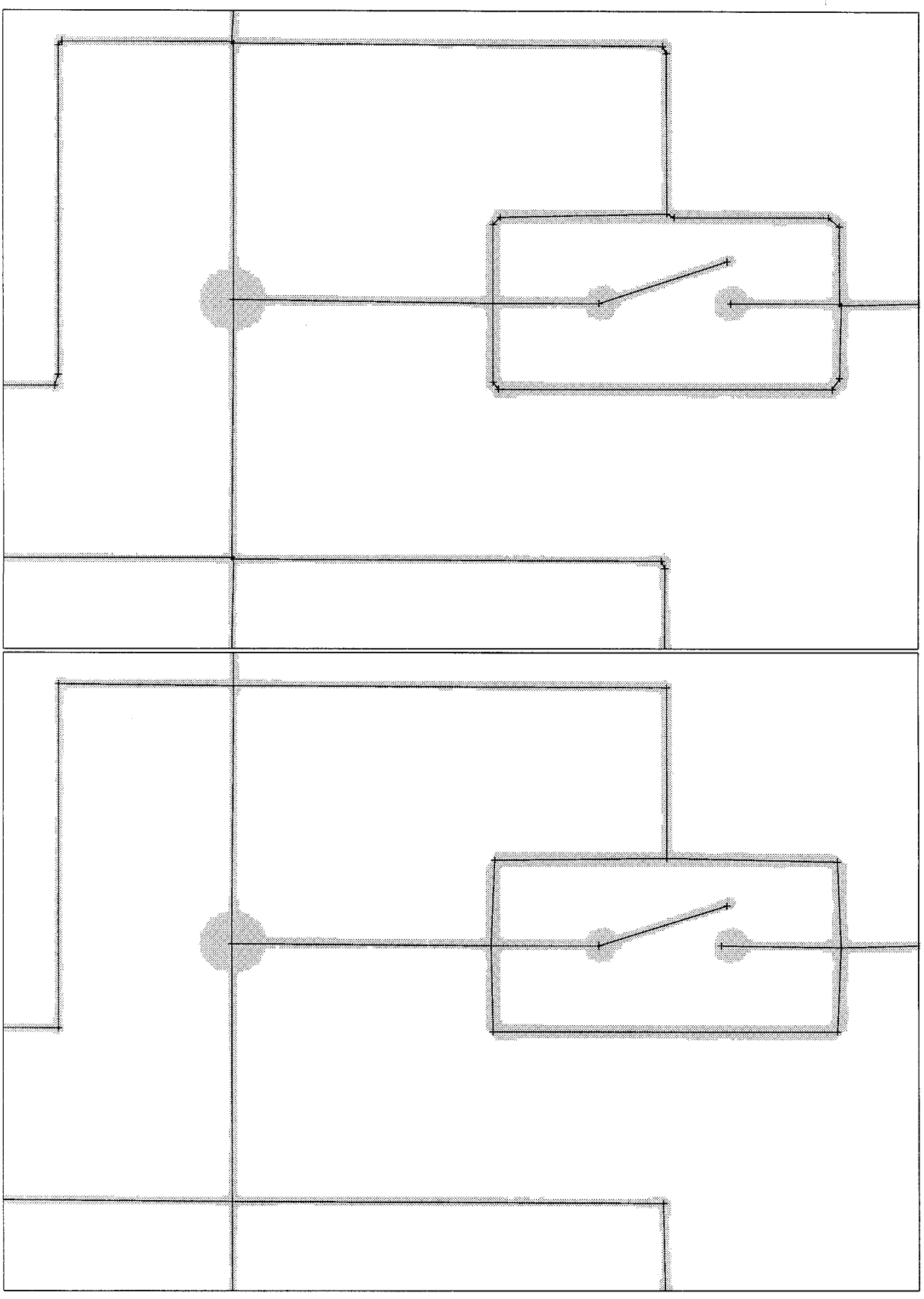


FIG. 11. Fraction of the results of experiment *scheme–dp* (top) and *scheme–clean* (bottom).

TABLE 3
Execution Time of the *dp* and *clean* Experiments, in Minutes and Seconds

	Execution time	
Map	*dp*	*clean*
geo	0′ 20″	4′ 04″
Leiden	1′ 13″	8′ 32″
gbkn	1′ 15″	11′ 37″
scheme	1′ 16″	15′ 22″

second for map geo, Leiden, and gbkn, and about 1 node per 1.7 second for line drawing scheme.

The problem with automatic vectorization is the determination of the ''significance'' of nodes. Some nodes can be moved over a larger distance than others while keeping their ''identity'' (i.e., the vectorization does not change significantly). Consequently, some nodes have more possible realizations than others. This can be associated with entropy. Examples of low-entropy nodes are dominant points, such as sharp corners, junctions or end points, whereas the nodes in a curve, or link nodes with a small bend angle between the edges have a high entropy. An accurate location of high-entropy nodes will be difficult to find because many ''correct'' locations are possible. Other solutions may be exploited: if many high-entropy nodes are found close together, they may be part of the same curve, and a curve segment may be constructed. No research has been done in this direction because the results of our algorithm (especially with the *clean* experiment) were already very satisfactory.

7. ROBUSTNESS AND IMPROVEMENTS

Influence of Global Line Thickness

Some parameters in the algorithm depend on the global line thickness w_{glob}. For example, nodes closer than w_{glob} pixels from an anchor point are merged with that point in the module which vectorizes between anchor points.

Two additional experiments were done to get an indication of the performance of the algorithm with different values of w_{glob}. Ideally, one should use the same line drawing, drawn with pens of varying thickness, but these drawings were not available. Scanning the same drawing with different resolutions would not work: in that case also the scale would be different. Therefore, we used the same drawing and used erosion and dilation to simulate drawings in which only the thickness of lines was different.

Map geo was eroded (experiment *geo–adap-eros*) and dilated (experiment *geo–adap-dil*) one 8-connected iteration, and experiment *geo–adap* was repeated. Both vectorizations were evaluated respectively on the eroded and dilated maps, and both on the original map (experiments *geo–adap-eros-orim* and *geo–adap-dil-orim*). The results of these experiments are given in Table 4.

Experiment *geo–adap-eros* shows a larger error than *geo–adap,* and this experiment in turn a larger error than *geo–adap-dil.* The first difference may be explained by the algorithm to compute the refined vectorization. There, nodes closer than w_{glob} pixels from an anchor node are merged with that node. In this experiment, that is a very short distance, and if the Douglas–Peucker algorithm fails to locate a node within the circle, the number of nodes and edges increases.

Experiment *geo–adap-eros-orim* indicates that the vectorization found is reasonable: the errors are lower than in experiment *geo–adap* but still higher than experiment *geo–dp.* The number of nodes and edges is highest for experiment *geo–dp,* and lowest for experiment *geo–adap.* Again, experiment *geo–adap-eros-orim* is in between. Sometimes, erosion can be a good idea before vectorization at the cost of more nodes and edges.

Experiment *geo–adap-dil* has a smaller error than *geo–adap-dil-orim.* This is so because our vectorization criterion is better satisfied in the former than in the latter experiment.

Influence of Scale or Resolution Change

The influence of scale or resolution change influences the algorithm only because lines may have a different global line thickness (assuming that lines will not be broken by a too low scanning resolution). This has already been discussed above.

Influence of ''Magic Numbers''

To avoid the use of too many ''magic numbers,'' most parameters have been made dependent on the global or local line thickness. The size of the structuring element is a certain factor (7) times the local line thickness. This and others have been chosen for efficiency: if the structuring element is larger, the cost for the computation of the morphological operations increases. Efficiency was the guideline for all magic numbers, except for the magic number $t_{\text{dp}} = 1\frac{1}{2}$ pixels: this value was chosen because it gives an accurate approximation of the lines in the original image. This can be seen from the experiments with extension *dp*: the results are good, at the cost of a larger number of nodes and edges.

Extensions to the Algorithm

The vectorization method described in this paper offers several possibilities for enhancement. The number of wrong lines can also be used for correcting the vectorization: if the wrong line is known, it can be corrected by

TABLE 4
The Results of the Experiments to Investigate the Influence of the Global Line Thickness

Map	w_{glob}	No. of pixels			No. of nodes	No. of edges	No. of lines of length				
		N_T	N_W	%			≥1	≥2	≥3	≥5	≥10
geo–dp	5.4	49,181	17	0.0	1845	1949	17	1	1	0	0
geo–adap	5.4	49,552	445	0.9	1613	1717	49	27	19	11	7
geo–adap-eros	2.9	49,982	2431	4.9	1768	1851	230	139	103	58	29
geo–adap-eros-orim	2.9	49,982	64	0.1	1768	1851	7	6	4	3	1
geo–adap-dil	7.8	49,393	38	0.1	1490	1587	16	6	4	3	0
geo–adap-dil-orim	7.8	49,393	901	1.8	1490	1587	182	125	75	27	13

moving one of its end points (this has been implemented in the experiments with extension *clean*), or by inserting a new node on the location of maximum deviation (this resembles the criterion used in the Douglas–Peucker algorithm).

Other kinds of (a priori) knowledge can be used. If it is known that certain line drawings have lines which always make an angle of multiples of 90°, our algorithm can be adapted easily to accommodate this property. Also, if it is known that most lines are parallel, that can be incorporated into the algorithm.

8. CONCLUSIONS AND SUMMARY

This paper presents a novel approach for vectorizing line drawing images. The method is based on a sequence of a standard vectorization algorithm and maximum threshold morphology, which can be iterated until a fitting criterion is met. Postprocessing modules can be added to handle specific properties of the line drawings.

In this paper, only the sequence (coarse) vectorization, morphological operation to correct nodes found, and final finer vectorization are used. Evaluation is done on different line drawings and results are discussed.

The best algorithm consisted of the following operations. The input image is skeletonized with a pseudo-Euclidian skeleton. After the determination of the global line thickness, the skeleton is vectorized with the Douglas–Peucker algorithm with a large threshold. For efficiency, link nodes whose edges make a small bend angle, and very short edges are removed. Then, from this coarse vectorization, the anchor points are determined using maximum threshold morphology. This is followed by a refined vectorization between the anchor points, which is postprocessed using two different rules. The first rule removes edges connected to nonanchor link nodes and reconnects the endpoints of these two edges with a longer edge. This is only done if the error according to the vectorization evaluation criterion is 0 for all edges involved. The second rule moves a node in the eight chain code directions under the condition that the error of the attached edges decreases. The new node must be an object pixel in the original bitmap. This is done first for nonanchor nodes, and then for anchor nodes. The application of these both rules is safe: using the first rule, the error cannot change, and using the second rule, the error can only decrease.

Our algorithm has several advantages over those mentioned in Section 3:

1. The algorithm is based on a sequence of a standard vectorization algorithm and maximum threshold morphology, which—in principle—can be iterated any number of times. However, we chose to use the sequence vectorization, morphology, vectorization because it provides a reasonable balance between a satisfactory result and reasonable processing time.

2. Due to the modular structure of the algorithm it can be adapted easily for special applications.

3. The results resemble those of interactive vectorization closer than some algorithms in Section 3, because first the gross outline of the vectorization is captured and then refinement among anchor points is performed.

The results of the algorithm in the experiments labeled *clean* are better than with the standard Douglas–Peucker algorithm. Also, our algorithm generates fewer nodes and edges than the Douglas–Peucker algorithm. However, the latter is faster.

The anchor points generated by our algorithm have a cartographic meaning because they give the gross outline of the line drawing. They may be useful in extracting features for an interpretation system.

ACKNOWLEDGMENTS

This research was supported by the Foundation for Computer Science in the Netherlands (SION), the Netherlands Organization for Scientific Research (NWO), and by the TopSpin project "Knowledge based conversion of utility maps (for the Provinciale Noordbrabantse Electriciteits Maatschappij PNEM)." The maps printed throughout this paper are reprinted with permission of the Dutch Cadastre. Research has been

conducted at the Delft University of Technology and the TNO Institute of Applied Physics, both in Delft, the Netherlands.

REFERENCES

1. M. Ejiri, S. Kakumoto, T. Miyatake, S. Shimada, and K. Iwamura, Automatic recognition of engineering drawings and maps. In *Image Analysis Applications* (R. Kasturi and M. M. Trivedi, Eds.), Chap. 3, pp. 73–126. Dekker, New York, 1990.
2. S. Suzuki and T. Yamada, MARIS: map recognition input system, *Pattern Recognit.* **23**(8), 1990, 919–933.
3. R. D. T. Janssen, R. P. W. Duin, and A. M. Vossepoel, Evaluation method for an automatic map interpretation system for cadastral maps, in *Proc. 2nd IAPR Int. Conf. on Document Analysis and Recognition, Tuskuba Science City, Japan, Oct. 20–22, 1993,* pp. 125–128. IEEE Comput. Soc. Press, 1993.
4. R. D. T. Janssen and A. M. Vossepoel, Compilation of mosaics from separately scanned line drawings, in *Proc. 2nd IEEE Workshop on Applications of Computer Vision, Sarasota, FL, Dec. 5–7, 1994,* pp. 36–43, IEEE Comput. Soc. Press, 1994.
5. G. M. Lammerts van Bueren, Scannen + vectoriseren of digitaliseren? Een vergelijking. *Geodesia* **35**(3), 1993, 114–118. [In Dutch]
6. A. J. Filipski and R. Flandrena, Automated conversion of engineering drawings to CAD form, *Proc. IEEE* **80**(7), 1992, 1195–1209.
7. D. Dori, Y. Liang, J. Dowell, and I. Chai, Sparse-pixel recognition of primitives in engineering drawings, *Machine Vision Appl.* **6**(2–3), 1993, 69–82.
8. K. Wall and P.-E. Danielsson, A fast sequential method for polygonal approximation of digitized curves, *Comput. Vision Graphics Image Process.* **28,** 1984, 220–227.
9. K. Wall, Curve fitting based on polygonal approximation, in *Proc. 8th Int. Conf. Patt. Rec., Paris, France, Oct. 27–31, 1986,* pp. 1273–1275, IEEE Comput. Soc. Press, 1986.
10. R. C. Gonzalez and R. E. Woods, *Digital Image Processing,* Addison–Wesley, Reading, MA, 1992.
11. C. M. Williams, An efficient algorithm for the piecewise linear approximation of planar curves, *Comput. Graphics Image Process.* **8,** 1978, 286–293.
12. C. M. Williams, Bounded straight-line approximation of digitized planar curves and lines, *Comput. Graphics Image Process.* **16,** 1981, 370–381.
13. J. Sklansky and V. Gonzalez, Fast polygonal approximation of digitized curves, *Pattern Recognit.* **12,** 1980, 327–331.
14. Y. Kurozumi and W. A. Davis, Polygonal approximation by the minimax method, *Comput. Graphics Image Process.* **19,** 1982, 248–264.
15. G. Dettori, An on-line algorithm for polygonal approximation of digitized plane curves, in *Proc. 6th Int. Conf. Patt. Rec., Munich, Oct. 19–22, 1982,* Vol. 2, pp. 739–741, 1982.
16. M. K. Leung and Y.-H. Yang, Dynamic two-stip algorithm in curve fitting, *Pattern Recognit.* **23**(1/2), 1990, 69–79.
17. T. Pavlidis, A hybrid vectorization algorithm, in *Proc. 7th Int. Conf. Patt. Rec., Montreal, Canada, July 30–Aug. 2, 1984,* pp. 490–492, 1984.
18. T. Pavlidis, A vectorizer and feature extractor for document recognition, *Comput. Vision Graphics Image Process.* **35,** 1986, 111–127.
19. A. Sirjani and G. R. Cross, An algorithm for polygonal approximation of a digital object, *Pattern Recognit. Lett.* **7**(5), 1988, 299–303.
20. C.-H. Teh and R. T. Chin, On the detection of dominant points on digital curves, *IEEE Trans. Pattern Anal. Machine Intell.* **11**(8), 1989, 859–872.
21. B. K. Ray and K. S. Ray, Detection of significant points and polygonal approximation of digitized curves, *Pattern Recognit. Lett.* **13**(6), 1992, 443–452.
22. D. H. Douglas and T. K. Peucker, Algorithms for the reduction of the number of points required to represent a digitized line or its caricature, *Can. Cartographer* **10**(2), 1973, 112–122.
23. J.-G. Leu and L. Chen, Polygonal approximation of 2-D shapes through boundary merging, *Pattern Recognit. Lett.* **7**(4), 1988, 231–238.
24. U. Ramer, An iterative procedure for the polygonal approximation of plane curves, *Comput. Graphics Image Process.* **1,** 1972, 244–256.
25. B. J. H. Verwer, Improved metrics in image processing applied to the Hilditch skeleton, in *Proc. 9th Int. Conf. Patt. Rec., Rome, Nov. 14–17, 1988,* pp. 137–142, IEEE Comput. Soc. Press, 1988.
26. C. J. Hilditch, Linear skeletons from square cupboards, in *Machine Intelligence 4* (B. Meltzer and D. Mitchie, Eds.), Chap. 22, pp. 403–420, Univ. Press Edinburgh, 1969.
27. L. R. Moore, Software for cartographic raster-to-vector conversion, in *Int. Archives of Photogrammetry and Remote Sensing, Proc. 17th ISPRS Congress, Washington, DC, Aug. 2–14, 1992,* 1992.
28. F. Attneave, Some informational aspects of visual perception, *Psychol. Rev.* **61**(3), 1954, 183–193.
29. R. van den Boomgaard, *Mathematical Morphology: Extensions towards Computer Vision,* Ph.D. thesis, University of Amsterdam, 1992.
30. A. Rosenfeld and J. L. Pfaltz, Distance functions on digital pictures, *Pattern Recognit.* **1,** 1968, 33–61.