Neural Comput & Applic (1998)7:343–355
© 1998 Springer-Verlag London Limited

Neural Computing & Applications

Shape Extraction: A Comparative Study Between Neural Network-Based and Conventional Techniques

A. Datta[1] and S.K. Parui[2]

[1]Computer and Statistical Service Centre, [2]Computer Vision and Pattern Recognition Unit, Indian Statistical Institute, Calcutta, India

Extraction of the skeletal shape of an elongated object is often required in object recognition and classification problems. Various techniques have so far been developed for this purpose. A comprehensive comparative study is carried out here between neural network-based and conventional techniques. The main problems with the conventional methods are noise sensitivity and rotation dependency. Most of the existing algorithms are sensitive to boundary noise and interior noise. Also, they are mostly rotation dependent, particularly if the angle of rotation is not a multiple of 90°. On the other hand, the neural network based technique discussed here is found to be highly robust in terms of boundary noise as well as interior noise. The neural method produces satisfactory results even for a very low (close to 1) Signal to Noise Ratio (SNR). The algorithm is also found to be efficient in terms of invariance under arbitrary rotations and data reduction. Moreover, unlike the conventional algorithms, it is grid independent. Finally, the neural technique is easily extendible to dot patterns and grey-level patterns also.

Keywords: Binary object; Dot pattern; Grey-level image; Medial axis; Neural network; Noise; Robustness; Rotation; Self organisation; Skeleton

1. Introduction

Shape description of objects is often necessary in image and document processing. This can be achieved by transforming the object to its skeleton which provides a representation of a pattern by a collection of lines, arcs or curves. The skeleton contains the essential structure of the object. This transformation process is called *thinning* or *skeletonisation*. Thus, thinning is the process of reducing the width of a pattern to just a single pixel. In processing *binary patterns*, thinning has been of continuous interest for the past three decades, and many algorithms have so far been suggested [1,2]. Skeletons are useful for topological analysis and classification of the shape of patterns containing elongated or linelike parts (for example, printed or handwritten characters, chromosomes, etc.). Character recognition systems often require thinning for extracting the basic features of the characters. Other advantages of thinning or skeletonisation are to reduce the memory space required to store the essential structural information of the pattern, and to simplify the data structure required in processing the pattern. Shape representation and data compression are the main objectives of skeletonisation, which plays an important role in image and document processing.

For digital binary objects, there are numerous definitions of a skeleton, and hence thinning algorithms differ from each other in performance and in implementation. But the main objective is to achieve a close approximation of the Medial Axis Transformation (MAT) of the object. The existing skeletonisation algorithms can be broadly classified into two categories – iterative algorithms and noniterative algorithms. The iterative algorithms remove outer layer pixels iteratively until a one-pixel thick-skeleton is achieved. According to the mode of removal of pixels, the iterative algorithms can be further classified into two classes – sequential and parallel algorithms. The sequential algorithms operate by

Correspondence and offprint requests to: Dr Swapan K. Parui, C. V. P. R. Unit, Indian Statistical Institute, 203 B. T. Road, Calcutta 700 035, India. Email: swapan@isical.ac.in processing the object pixels in a sequential manner, while in parallel algorithms all or a subset of the pixels are processed in parallel. The parallel algorithms are of two types – multi-pass and single-pass. In the noniterative class, various techniques are used (for example, distance transform, polygonal approximation, contour following, etc.). This class of algorithms produces a certain skeleton of the pattern directly in one pass without examining all the individual pixels. The output skeleton here may not be in the form of a raster image. It may be in the form of a planar graph providing a line-segment approximation of the input pattern. Henceforth, we shall call these two types of skeleton a *raster skeleton* and a *vector skeleton*, repectively.

In the present article, we focus on the performance evaluation of a new approach to skeletonisation that satisfies all the above properties. The new approach is an improvement upon a self-organising neural network model [3] for the purpose of skeletonisation. The earlier algorithm was sensitive to the choice of parameter values, and the results obtained were often unsatisfactory for skeletonisation. The present algorithm based on the new concepts of 'activation level' and 'activation region', which are data dependent, is sufficiently adaptive and is more efficient in skeletonisation. Also, it has been examined, in a comprehensive manner, how the improved algorithm performs in terms of several important aspects which are relevant to skeletonisation.

Self-organisation is a learning phenomenon that has been observed in the human neural system. It is simulated with an artificial neural network model by Kohonen [4]. Kohonen's Self-Organising Neural Network (KSONN) model is essentially a feature mapping which has some interesting properties. For example, it preserves the topological properties of the input, and can select the dimensionality automatically. The present skeletonisation algorithm uses the fundamental rules and properties of KSONN model, but does not use it in its original form. The model has been modified to make it dynamic to fit it into the skeletonisation task. Different dynamic versions of the KSONN model have been reported by several authors in different contexts [5–7]. Our model produces a vector skeleton in the form of a planar graph.

The performance of an algorithm should be judged not only by measuring how well it works on perfect input data, but also by measuring how it works on noisy data. An algorithm may give good results with noise-free input, but very poor results in the presence of even a little amount of noise. Such algorithms are not of much use. Since noise is almost unavoidable in practice, more robust algorithms are needed. Here, we first describe our model and algorithm for skeletonisation of a noise-free binary image. The image consists of two levels – black for object pixels and white for background pixels. We then add random noise (black noise in background and white noise in object) to the image with different SNR values, and study the difference among the resulting skeletons. The proposed algorithm is qualitatively different from the conventional skeletonisation algorithms in terms of noise sensitivity. In conventional algorithms, the noise type that is usually considered is boundary noise only. In the present study, both the boundary and object noise types are taken care of. It can be seen that the proposed algorithm is highly robust in terms of both these types, while the conventional algorithms are not designed to tackle object noise.

We describe in brief, before going into the actual comparisons, our neural network model for skeletonisation [3] in Section 2. Section 3 presents the comparative studies, with the help of some examples, between the neural algorithm and some conventional thinning algorithms. The comparative studies are done on the basis of the following properties of a good skeletonisation algorithm: (a) noise immunity or robustness; (b) rotation invariance; (c) close approximation of the medial axis; (d) high data reduction efficiency; (e) extendibility to other input types – *dot patterns* and *grey-level patterns*. These properties are dealt with in different subsections. Conclusions are given in Section 4.

2. Network Model and Algorithm

There are a few shortcomings of Kohonen's SONN model in view of the skeletal shape extraction of a pattern. In Kohonen's SONN model, a network having either a linear topology or a planar topology is used. In general, such rigid neighbourhood topologies are found unsuitable in some situations and, in particular, it poses problems in skeletal shape extraction of a pattern. When the input pattern has a prominent shape such neighbourhood definitions are found unsuitable [8]. This is due to the fact that, during the update process, the weight vectors lying in zero-density areas may be affected by input vectors from the surrounding parts of the nonzero distribution. As the neighbourhoods are shrunk, the fluctuation vanishes and, as a result, some processors may remain outlier due to the residual effect from the rigid neighbourhood. For example, if the input vectors have a circular distribution, the processors near the centre are not representative of the input.

Another problem with the SONN model is that it assumes a predefined fixed topology of the network which is maintained throughout. But in skeletonisation tasks, at different localities of the input pattern, we require a varying topology of the network. For example, at different portions of the pattern, it may have different structures namely, an arc, a fork, a crossing, etc. Thus, a linear chain of processors fails to adapt the skeletal shape of circular and fork structures. For a circular pattern, in the SONN model, the topology of the pattern is no longer preserved by the output network (the pattern is a closed loop while the output network is open). The fork structure cannot be adapted by a linear network structure. To adapt the fork structure, the network requires one processor to have three neighbours. To overcome the limitations of Kohonen's model, we suggest some modifications [3] of it in which the set of processors and their neighbourhoods change adaptively during learning, in order to extract the skeleton of a binary image.

Kohonen's feature mapping net is composed of an array A (1- or 2-dimensional) of processors (neurons) receiving input signals from a feature space V to be mapped onto A. Each input vector is presented to the net on m input lines, where m is the dimension of the space V. Each processor in the array is connected to one or more surrounding processors. Every processor is also connected to all input lines. The map is adapted on the basis of a set of input vectors from the feature space.

Denote the set of processors by $\{\pi_1,\pi_2,\ldots,\pi_n\}$. The *neighbourhood* N_i of a processor π_i is $\{\pi_p|\pi_p$ is connected to $\pi_i\}$, which includes π_i. Let the weight vector for the processor π_i be $W_i(t) = (w_{i1}(t),w_{i2}(t),\ldots,w_{im}(t))$ at the time instance t. The starting weight vectors $W_i(0)$ are chosen at random. Suppose the set of input vectors is $S=\{P_1,P_2,\ldots,P_N\}$, where the dimension of each P_j is m. The weight vectors are updated according to the following rule.

At time instance t, P_j is presented to the net. All the processors compete and let $W_k(t)$ be the nearest weight vector to P_j. That is,

$$|W_k(t) - P_j| = \min_i |W_i(t) - P_j| \tag{1}$$

Then, the weight vectors of the processors within the neighbourhood of π_k are updated as [4]:

$$W_p(t+1) = W_p(t) + \alpha(t)[P_j - W_p(t)], \quad \text{for } \pi_p \in N_k,\ 0 < \alpha(t) < 1 \tag{2}$$

where $\alpha(t)$ is the gain term which decreases with t.

We have classified the input binary patterns into three categories: (1) Arc patterns like character patterns 'C', 'L', 'S', 'M', etc. which have a linear structure; (2) tree patterns like 'T', 'X', 'Y', etc. which have forks and branchings; (3) loop patterns like 'A', 'B', 'O', etc. which contain a loop structure. These categories are dealt with separately.

2.1. Arc Patterns

The structure of these patterns can be represented by a linear structure. The initial net having a linear structure is represented by a list of processors $[\pi_1,\pi_2,\ldots,\pi_n]$, where π_i is connected to exactly two processors π_{i-1} and π_{i+1} (the two end processors are connected to exactly one processor each). Here, the input feature vectors are the co-ordinates of the object (black) pixels of the image, and hence $m=2$. $S=\{P_1,P_2,\ldots,P_N\}$ is a set of N pixels, where $P_j=(x_j,y_j)$. The weight vectors of the processors π_i are updated iteratively on the basis of the pixels in S. The initial weight vectors of π_i are, say, $(w_{i1}(0),w_{i2}(0))$. Suppose, the pixel P_j is presented at the tth iteration. Let

$$dist(P_j,W_k(t)) = \min_i [dist(P_j,W_i(t))].$$

P_j updates the weight vectors according to Eq. (2).

If this updating process continues, the weights tend to approximate the distribution of input vectors in an orderly fashion. The limiting weight vectors define the ordering. One presentation each of all the pixels in S makes on *sweep* consisting of N iterations. After one sweep is completed, the iterative process for the next sweep starts again from P_1 through P_N. Several sweeps make one *phase*. One phase is completed when the weight vectors of the current set of processors converge, i.e. when

$$|W_i(t) - W_i(t')| < \varepsilon \quad \forall i \tag{3}$$

where t and t' are the iteration numbers at the end of two consecutive sweeps and ε is a predetermined small positive quantity. Only after a phase is completed are processors inserted or deleted. Suppose, at the end of the sth phase, the weight vectors of the processors are $W_1(t_s),\ldots,W_{n(s)}(t_s)$, where $n(s)$ is the number of processors during the sth phase and t_s is the total number of iterations needed to reach the end of the sth phase. If the weight vectors of two neighbouring processors now become very close, the processors are merged. If their weight vectors are far apart, a processor is inserted between them. More formally, if

$$|W_k(t_s) - W_{k'}(t_s)| = \min_{i=1,\ldots,n(s)} \min_{\pi_{i'}\in Ni-\{\pi_i\}} |W_i(t_s) - W_{i'}(t_s)| < \delta_1 \tag{4}$$

then the two processors π_k and $\pi_{k'}$ are merged, and the new processor has the weight vector as $[W_k(t_s) + W_{k'}(t_s)]/2$. If, on the other hand,

$$|W_l(t_s) - W_{l'}(t_s)| = \max_{i=1,\ldots,n(s)} \max_{\pi_{i'} \in N_i - \{\pi_i\}} |W_i(t_s) - W_{i'}(t_s)| > \delta_2 \quad (5)$$

then one processor is inserted between π_l and $\pi_{l'}$ and the new processor has the weight vector as $[W_l(t_s) + W_{l'}(t_s)]/2$. Note that δ_1 and δ_2 are two predetermined positive quantities such that $\delta_1 < \delta_2$. After the insertion and merging of processors, the next phase starts with the new set of processors. The process continues until, at the end of a phase,

$$\text{for all } i, \quad \delta_2 > |W_i(t) - W_{i'}(t)| > \delta_1, \quad \forall\ \pi_{i'} \in N_i - \{\pi_i\} \quad (6)$$

The condition (6) means that the weight vectors of no two neighbouring processors are either two close or too far apart. The distance between two neighbouring processors (after convergence) is controlled by δ_1 and δ_2. The processors (on the basis of their weight vectors) at this stage give an approximate skeletal shape of the input pattern. It should be noted that the quality of the skeleton depends on the choice of δ_1 and δ_2. This issue is discussed in more detail in Section 2.4.

2.2. Tree-like Patterns

For tree-like patterns, it is required that some processors in the net have more than two neighbours. The *degree* of a processor is defined as the number of its neighbours. Unlike in the case of arc patterns, the degree for each processor should be learned here.

Let us consider a pattern with a fork. Since initially there is no topological information about the pattern, we start with a linear net with five processors (Fig. 1a). After a number of iterations, some processor forms a significantly small acute angle (decided on the basis of some threshold) with its two neighbours (Fig. 1c) to indicate a fork in the pattern. Such a spike is formed because, by a property of Kohonen's feature map, the net tries to span the entire range of input pattern space, and also the topological relationship of the pattern is preserved in the net. In Fig. 1(c), processor X forms a spike with its neighbouring processors Y and Z, indicating a juncture lying between Y and Z. Therefore, the following actions are taken to adapt the degree (Fig. 1d) when processor X forms a spike:

(a) Create a new processor say, U (denoted by a solid circle) halfway between Y and Z.
(b) Delete the link between X, Y and the link between X, Z.

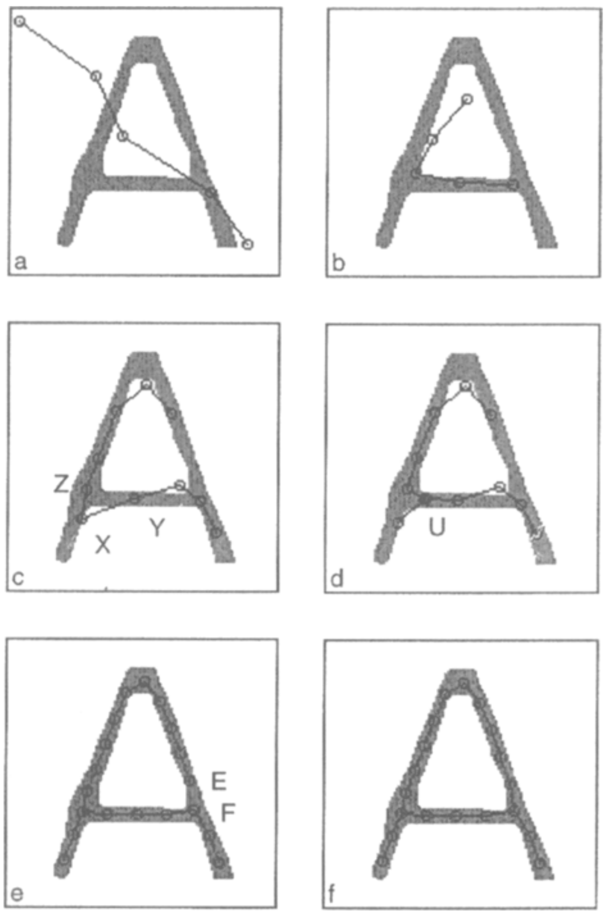

Fig. 1. Different steps of convergence of the net for pattern 'A' without noise. (a) initial net, (b) after 12 sweeps, (c) formation of an acute angle after 94 sweeps, (d) after 105 sweeps with a new processor (solid circle) created with degree = 3, (e) after 394 sweeps immediately before loop formation, (f) final net after 518 sweeps. ($\delta_1 = 6$, $\delta_2 = 12$ and $\varepsilon = 0.01$.).

(c) Establish links between U, X; between U, Y and between U, Z.

The same actions are taken for all the processors forming a spike. These actions are taken after a phase is complete, and then the subsequent phases of learning are continued to enable the net to approach towards a closer approximation of the shape of the pattern. Similar principles used in the case of arc patterns are followed for insertion and deletion of processors, and for convergence of the algorithm.

2.3. Loop Patterns

The techniques discussed above work for patterns excepting those containing loops. Consider the pattern 'A'. Our algorithm can generate, on the basis of the principles discussed in Section 2.2, an incomplete skeleton, as shown in Fig. 1(e). It is now necessary to complete the loop by means of bridging the gap (between processors E and F in Fig. 1e).

The asymptotic values of the weight vectors constitute some kind of vector quantisation [4]. In particular, the distance measure and the update rules as considered in our algorithm, induce a partition of the input pattern space specified as

$$S_i = \{P_j \in S | dist(P_j, W_i) \leq dist(P_j, W_r) \; \forall r\}$$

The above partition is a Voronoi tessellation which, in the present situation, means partitioning of the input pattern space into regions within each of which all input vectors have the same weight vector as their nearest one. Therefore, each set S_i is associated with a single processor. Hence the input pattern vectors can easily be labelled according to the S_i to which it belongs. In other words, each input vector is given a label according to its nearest processor. Such input labelling has earlier been used by Sabourin and Mitiche [5].

Definition. When the input pattern is a binary image, two processors π_i and π_j $(i \neq j)$ are said to be *adjacent* if there exists at least one pair of object pixels $P \in S_i$ and $Q \in S_j$ such that P and Q are 8-neighbours of each other.

After convergence (let us call it initial convergence) as mentioned in Sections 2.1 and 2.2, label the input vectors as mentioned above and then for each processor, check whether it is adjacent to any processor other than its neighbours. If it is, introduce a link between these two processors. In Fig. 1(e), processor E is adjacent to processor F, and they are not neighbours to each other. So, they are connected and become neighbours. After this we continue the algorithm until the final convergence is reached (Fig. 1f), i.e. when condition (6) is satisfied. The algorithm can now be stated briefly as:

The Algorithm

```
Step 1:  [Initialisation]
         Initialise t = 0;
         Initialise the weight vectors $W_i(t)$,
         $(i = 1,2,\ldots,n)$ with random values.
Step 2:  [Sweep]
         For all input patterns $P_j$, $j = 1,2,\ldots,N$
         Update weight vectors according to rule (2).
Step 3:  [Phase]
         If condition (3) is not true then go to Step 2.
Step 4:  Merge or insert according to condition (4) or (5).
Step 5:  If condition (6) is not true go to Step 2.
Step 6:  If no processor forms a spike go to Step 9.
Step 7:  Create a new processor U, as mentioned in Section 2.2.
Step 8:  If condition (6) is not true go to Step 2.
Step 9:  Label the input vectors as mentioned above.
         If no processor is adjacent to any processor
         other than its neighbour then Stop.
Step 10: Join the processor and the processor adjacent to it.
Step 11: If condition (6) is not true go to Step 2.
Step 12: Go to Step 6.
```

The final network obtained by the above algorithm gives a vector skeleton for the given input pattern. The raster skeleton can easily be derived from the network. For each link, the line segment connecting the weight vectors of the two corresponding processors is considered. The set of all pixels lying closest to such a line segment gives the raster skeleton.

The above algorithm has been tested on a number of input patterns. The final results along with a few intermediate stages are shown in Fig. 2 for character patterns ‘S’, ‘X’, ‘a’ and ‘y’. The initial nets for all of them are the same as shown in Fig. 1(a).

For arc patterns, it can easily be seen that the resulting net, after convergence, gives a skeletal shape of the pattern. Here the array of processors is linear, and the inputs are from a two-dimensional distribution (see [4, p.153]). In the present model, we start with a given number of processors and, at the end of a phase, we obtain an output similar to the Kohonen’s model. After each phase, either a processor is deleted or two consecutive processors are merged into a single one. In the process of insertion and merging, the existing global ordering of the processors is never disturbed. Note that each phase here can be looked upon as a full execution of Kohonen’s algorithm, because each phase starts with a given number of processors and converges to an output. Thus, the only difference between our model and the original model in terms of convergence is that the former is a repetitive application of the latter, each time increasing/reducing the size of the net without disturbing the global ordering. From the fact that once the processors are ordered they remain so for all t (see [4, p.143]), the output net in the proposed model will give the skeletal shape of the pattern, as given by Kohonen’s model.

For tree-patterns, after a few phases (when almost all the weight vectors are positioned within the pattern, or at least quite close to it), a spike in the net is replaced by a ‘Y’- or ‘T’-like structure locally. The starting net being linear, a spike here represents

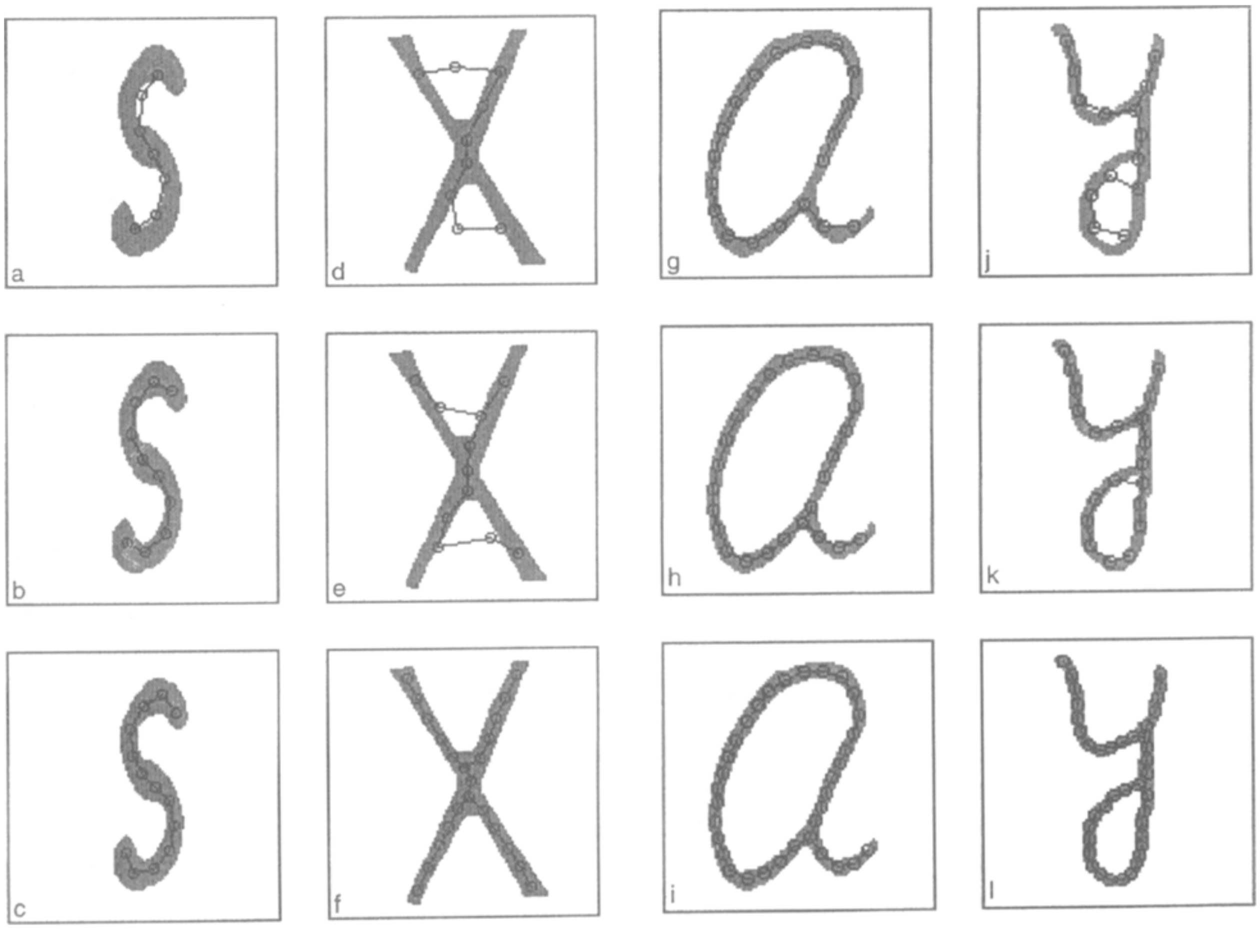

Fig. 2. Two intermediate steps and the final skeletons for patterns 'S', 'X', 'a' and 'y'.

a junction in the local neighbourhood of the pattern. The method is repeated after each phase. For a '+'-like junction, two such replacements are required. For other parts of the pattern, the argument holds good, since a tree-pattern is a union of several arc-patterns.

If the pattern has a loop, the algorithm (up to Step 8) yields only a tree-structured net. Subsequently, loops are formed depending on the Voronoi regions. If two Voronoi regions are adjacent but the respective processors are not already linked, then a link between the two processors is established.

2.4. Choice of δ_1 and δ_2

It is easy to see that δ_1 and δ_2 play a role here. Very low values of δ_1 and δ_2 might produce a zig-zag (peano curve) network (Fig. 3a–b) which does not represent the true skeletal shape of the pattern. On the other hand, if δ_1 and δ_2 are very high, the skeleton does not properly represent the medial axis (Fig. 3e–f). Experiments have shown that the skeleton starts becoming zig-zag when δ_2 (with $\delta_1 \approx \frac{1}{2}\delta_2$) becomes less than $\frac{1}{3}T$ where T is the local thickness of the pattern. Values of δ_2 larger than $\frac{1}{3}T$ can prevent the zig-zagness. Thus, if the input pattern has more or less uniform width and the approximate width value is known, then the choice of δ_1 and δ_2 is easy. But in practical situations, the thickness may not be uniform or, even if it is, its value may not be known. In such situations, some adaptive mechanism is necessary. One such mechanism is to introduce an *activation level* in the weight updating process, which is described below.

We specify an *activation region* of a processor so that if an input vector falls within the region then only it activates the processor. The activation region decreases over time. In the present problem, it is taken as the circle with the corresponding weight vector as the centre and the activation level as its radius. The values of the activation level $a_i(s)$ for $i,\dots,n(s)$, at the end of the sth phase, are computed as follows:

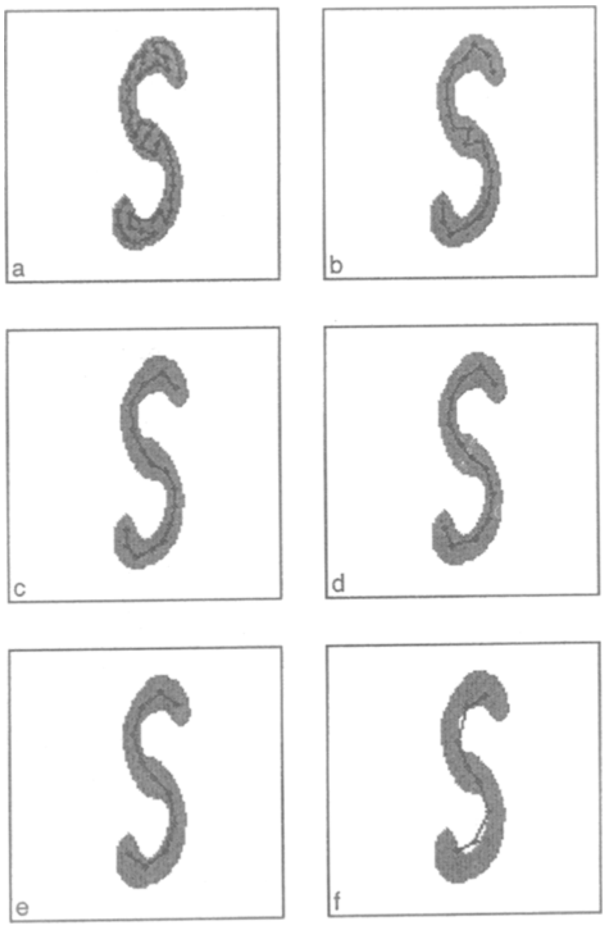

Fig. 3. The role of δ_1 and δ_2 on the output skeleton. (a) $\delta_1 = 3$, $\delta_2 = 6$, (b) $\delta_1 = 4$, $\delta_2 = 8$, (c) $\delta_1 = 5$, $\delta_2 = 10$, (d) $\delta_1 = 6$, $\delta_2 = 12$, (e) $\delta_1 = 7$, $\delta_2 = 14$, (f) $\delta_1 = 8$, $\delta_2 = 16$.

$$a_i(s) = \max_{\pi_{i'} \in Ni - \{\pi_i\}} |W_i(t_s) - W_{i'}(t_s)| \tag{7}$$

A processor is called *active* if the presented input vector lies within its activation region. In other words, if an input vector lies outside all the activation regions, it is ignored in the competition and updating process (Eqs 1 and 2). Only active processors are qualified for the competition, after which the winner processor is selected. As the activation region is shrunk over time (according to Eq. 7), the object pixels away from the medial axis gradually and symmetrically loose their effects on the weight adjustments. Thus, all the weight vectors approach the medial axis. The algorithm described earlier is modified to incorporate the activation level. Results of the modified algorithm are shown in Fig. 4. It is found that the modified algorithm gives much better results. In the original algorithm, if δ_1 and δ_2 take small values (say, $\delta_1 = 1$, $\delta_2 = 2$) then the skeletons become zig-zag (Figs 4a, b), but in the modified algorithm, the skeletons do not become so with the same δ_1 and δ_2 (Figs 4c, d). Thus, we can always set $\delta_1 = 1$ and $\delta_2 = 2$ and, by introducing the activation level, a satisfactory medial axis representation can be obtained.

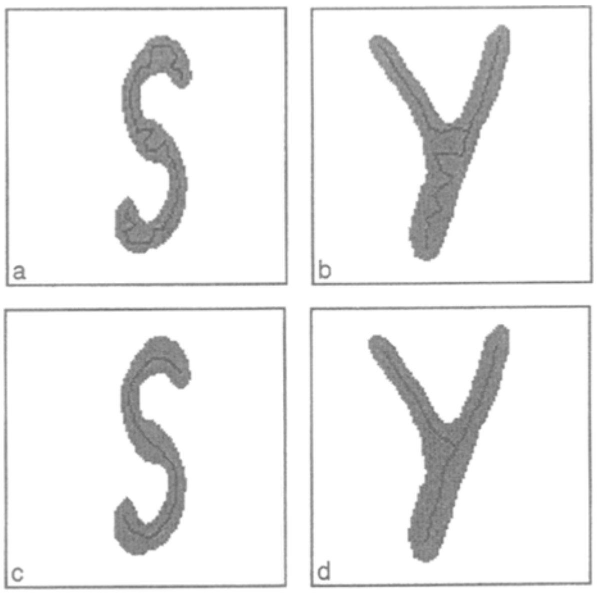

Fig. 4. The raster skeletons obtained (a), (b) without and (c), (d) with activation level. $\delta_1 = 1$, $\delta_2 = 2$.

Note that smaller values of δ_1 and δ_2 will take a longer time to converge. However, in some applications, a crude approximation of the skeletal shape serves the purpose. In such situations, we can choose higher values of these two parameters and get the output more quickly. Thus, the user has an option to make the algorithm faster at the cost of accuracy.

3. Comparisons Between Neural and Conventional Techniques

3.1. Robustness

An important aspect of a skeletonisation algorithm is noise immunity, which makes the present algorithm qualitatively different from the conventional ones. Two types of noise, namely boundary noise and object noise, are considered here. If the original object contains noise, the skeleton should not deviate much from the skeleton of the same object without noise. A serious problem with most of the existing algorithms is that they sometimes produce noisy skeletons if the input patterns contain noisy boundary (see Datta and Parui [9] and Jang and Chin [10]). Moreover, these algorithms cannot handle object noise. On the contrary, the proposed algorithm is designed to take care of both these types of noise and is highly robust to them.

The above claim of robustness of the present algorithm can be argued as follows. The resulting skeleton here is given by the weight vectors after convergence, and their links. Its final position is highly insensitive to noise pixels because of two factors. First, the weight vector converges to the centre of gravity of the respective Voronoi region (S_i) and this centre is not greatly affected by noise pixels. Secondly, the activation region of a processor decreases over time and, as a result, the boundary noise pixels are kept outside it, to a great extent. Thus, the noise insensitivity of the present algorithm is clear from its learning mechanism and convergence property.

Most existing conventional algorithms use a rigid definition of connectedness of the object – which in effect causes noise sensitivity. Our method relaxes the concept of connectivity, and it is found that such a relaxation helps the robustness, particularly in situations where SNR is very close to 1.

3.1.1. Boundary Noise. The boundary noise is distributed on the boundary of the object (white noise) and on its immediate neighbourhood in the background (black noise). The proposed skeletonisation algorithm is found to be very robust to such boundary noise. Here we add black and white boundary noise pixels and study the effect they have on the skeleton. We have experimented on several examples with different SNR values, where the SNR is defined as follows:

$$SNR_B = \frac{\text{Number of boundary black pixels}}{(B+W)} \tag{8}$$

where B = number of black noise pixels and W = number of white noise pixels. To demonstrate the effect of boundary noise, results are presented for a straight line pattern of length 33 pixels and width 5 pixels, where '0' represents an object pixel and '*' represents a skeletal pixel (Fig. 5). The skeletons obtained by the present algorithm, and by several conventional algorithms [9,11–14] are shown in Figs 5(b)–(e) for a fixed SNR = 18. There is no distortion in the output skeleton obtained by our algorithm. For different SNR values, the output skeletons obtained by our algorithm from the same pattern are given in Figs 5(f)–(i). The output skeleton remains undistorted even with SNR = 3. At SNR = 2, the output skeleton has some distortion with $m_e = 0.31$, where (as in Jang and Chin [10])

$$m_e = \min\left\{1, \frac{Area[S_K - S'_K] + Area[S'_K - S_K]}{Area[S_K]}\right\} \tag{9}$$

where S_K and S'_K are the skeletons obtained from S and its noisy version, respectively. Area [·] is an operator that counts the number of pixels.

The proposed algorithm has been tested on 62 test patterns (English upper and lower case characters and numerals), and the average values of m_e against different SNR values are computed. Using the same measure, Jang and Chin's [10] algorithm has been shown to be superior to several other conventional algorithms. The average m_e values, in our algorithm, are found to be less than 0.04 for SNR values ranging from 20 to 3. For SNR = 2, the average value of $m_e \approx 0.35$. In Jang and Chin's algorithm, even for SNR = 10, the average value of m_e is higher than 0.2; and for SNR = 20, it is higher than 0.1. For SNR values higher than 20, our algorithm produces m_e that is practically zero. Thus, it is found that our algorithm is even better than Jang and Chin's algorithm in terms of boundary noise immunity.

3.1.2. Object Noise. By object noise we mean the white noise distributed over the entire object, including its boundary. The existing conventional algorithms are not able to handle such noise, which is interior to the object, but such noise may occur in practice for several reasons. The problem with conventional algorithms (for example, the iterative ones) are that they use the property of local connectivity within a small window (mostly 3 × 3), and try to preserve such local connectivities throughout. Secondly, due to the use of a fixed set of templates of a small size, they treat a single white noise pixel as a hole consisting of a single pixel. As a result, in the output skeleton it appears as a big hole (Fig. 6(b)). Figure 6 demonstrates how only two noise pixels misclassify a '1'-like pattern to an '8'-like pattern. On the contrary, the proposed algorithm uses the connectivity concept in a more global sense (for example, while joining two processors, we check whether the two respective regions are adjacent). The algorithm treats small holes as white noise at the cost of the possibility of missing a true small hole. Thus, very small holes have hardly any effect on the resulting skeleton (Fig. 6(c)). But if the hole is large enough and is in fact a part of the pattern, it is output as a hole in the resulting skeleton (Figure 1(f)). We have experimented, and found that the algorithm is robust and performs satisfactorily with moderately low SNR (moderate amount of noise), where the SNR is defined by

$$SNR = \frac{\text{Number of object pixels}}{\text{Number of noise pixels on the object}} \tag{10}$$

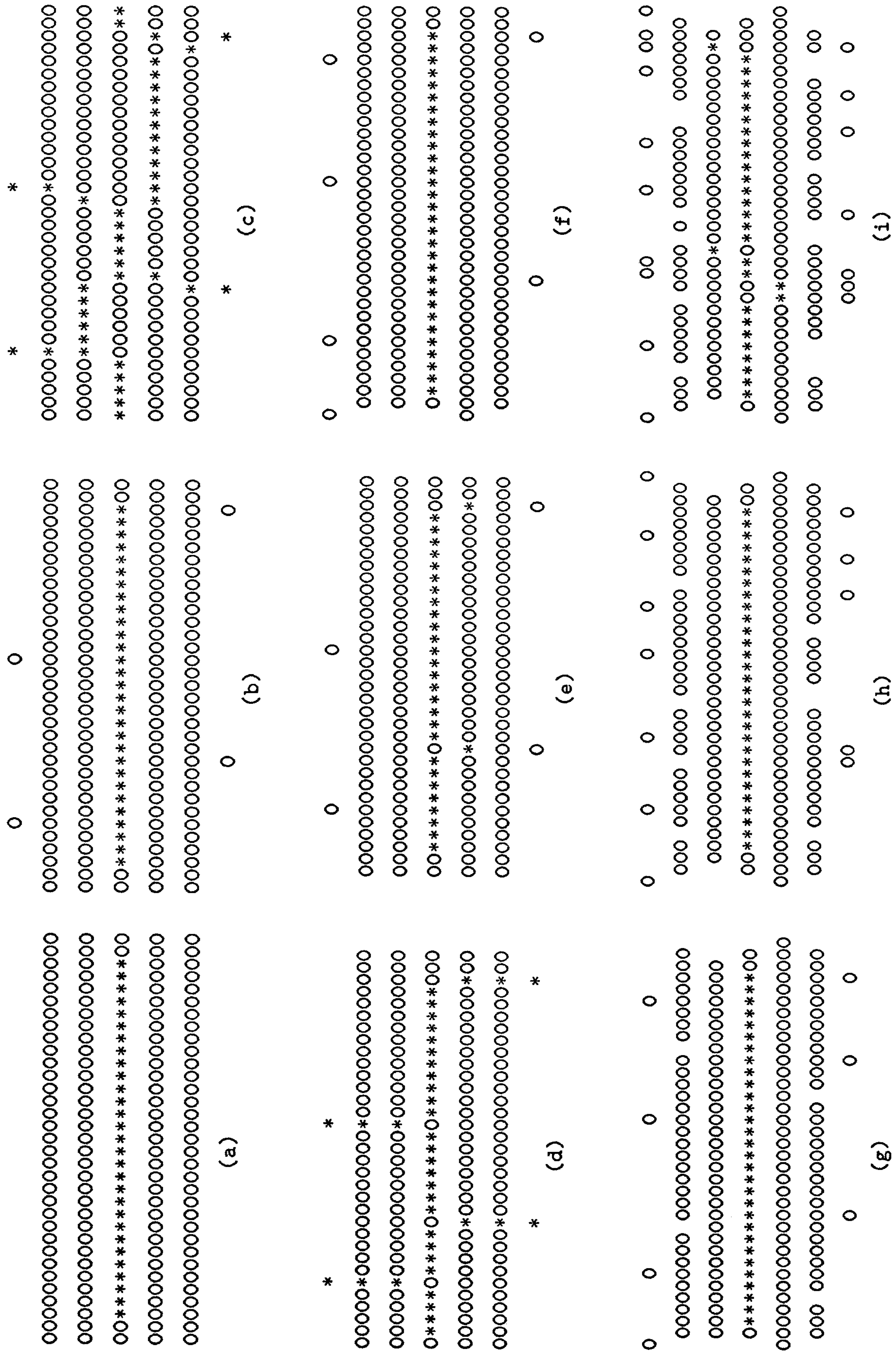
Fig. 5. '0' means object and '*' means skeleton. (a) Raster skeleton obtained by our algorithm. The same with SNR = 18 (b) by our algorithm, (c) by [11,12,13], (d) by [14], (e) by [9]; (f)–(i) skeletons by our algorithm with approximate SNR values as 10, 5, 3 and 2, respectively.

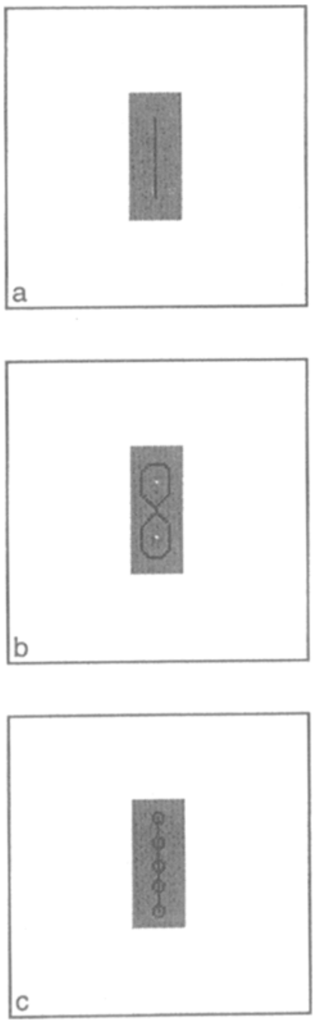

Fig. 6. Output skeleton generates big holes in the presence of even single noise pixels. Output of conventional thinning algorithms (a) without any noise, (b) with two single noise pixels, (c) output of the proposed algorithm with the same noise as in (b).

For a very low SNR, i.e. for a very high amount of noise, when the binary object merely becomes a set of scattered pixels or a dot pattern, the proposed algorithm with some modifications in the loop joining step can still yield the global skeletal shape of the pattern, assuming the noise to be uniformly distributed.

It is clear that the loop joining process, by checking the adjacency of two respective regions' S_i's, is meaningful as long as the connectivity of the object is preserved after noise addition. Otherwise, the loop joining has to be done in a different way. In fact, by the properties of Kohonen's model, two processors will be neighbours to each other if the two respective Voronoi regions are close in the pattern space, since Kohonen's self-organising map preserves the topological relationship. Hence in our case, it is expected that two close processors should be neighbours to each other. Therefore, we join two processors by a link if they are close enough, but are not already joined. The closeness is determined on the basis of δ_2. Formally, loop joining step can be stated as follows:

> For every processor, its nearest among other processors (excepting its neighbours) is found. If the distance between these two processors is less than δ_2, then they are joined by a new link.

Thus, after the initial convergence, we join the processors satisfying the above criteria. Then we continue the algorithm until final convergence is reached, i.e. until condition (6) is satisfied.

An illustration is presented in Figs 7(b)–(d) for the pattern 'A' with different SNR values. We have taken a very high amount of object noise and tested the algorithm for several character patterns. It has been found that even in the presence of very high noise (SNR = 2.0, 1.5 and 1.1), the proposed algorithm is able to extract the skeletal shape of the original object as can be seen in the example figure (Fig. 7). The existing conventional algorithms fail to work in such situations.

3.2. Rotation Invariance

Another advantage of our algorithm is that the output skeleton does not depend upon rotation of the input pattern by arbitrary angles. This is because the proposed method does not assume any underly-

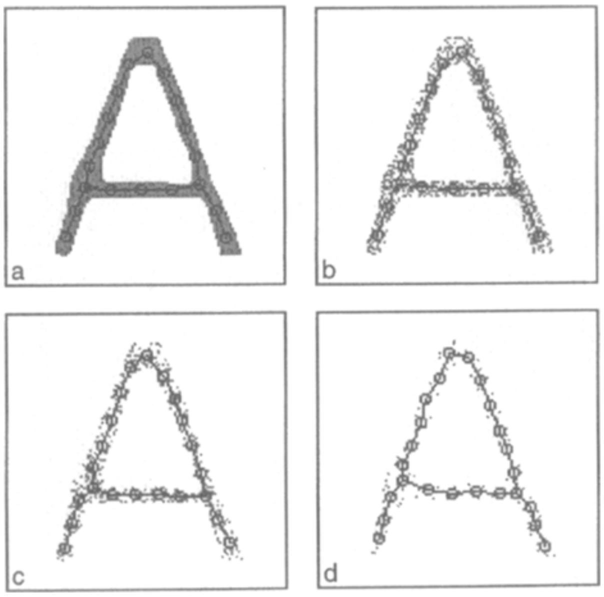

Fig. 7. The final skeletons obtained for the pattern 'A' with object noise. (a) Original binary pattern, (b) SNR = 2, (c) SNR = 1.5, (d) SNR = 1.1.

ing grid. The iterative methods for skeletonisation that use square grid are invariant under rotation by multiples of 90° only. A recent paper [10], based on derived grid, reports invariance under 45° rotation also. As an illustrative example, we have rotated the pattern 'A' by different angles (11°, 22° and 45°) and shown the output skeletons in Fig. 8. Figures 8(b)–(d) show almost no effect of rotation of the input pattern on the output skeleton.

3.3. Medial Axis Representation

As the basic purpose of skeletonisation is to approximate the medial axis of the object pattern, it is important that the output skeleton should approximate the medial axis as closely as possible. In other words, the skeleton should not be biased. After getting the raster skeleton, the following measure [14] is computed for comparison of the goodness of medial axis representation with some other thinning algorithms:

$$\eta = \frac{Area[S'']}{Area[S]} \tag{11}$$

where S = set of all object pixels in the input pattern, and S'' = union of the maximal digital disks (included in S) centred at all the skeletal pixels. Clearly, η ranges from 0 to 1, and the derived skeleton is identical to the ideal medial axis if η is 1. η measures the closeness of the extracted skeleton to the true medial axis. Jang and Chin [14] found the average values of η to be 0.712 and 0.881 on a number of patterns for the algorithms in Arcelli; et al. [11], Tamura [12] and Hilditch [13] and the algorithm in Jang and Chin [14], respectively. In Datta and Parui [9], this measure was further improved to 0.931. For the proposed algorithm, the average η value is found to be 0.885 (with $\delta_1 = 1$ and $\delta_2 = 2$) for the same set of test patterns.

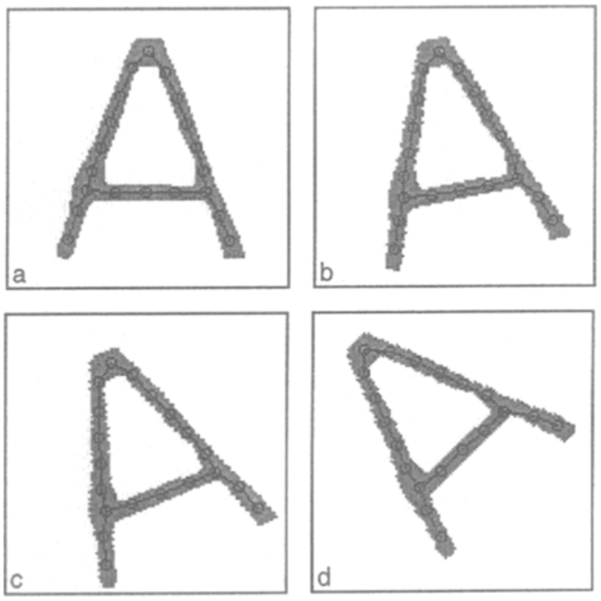

Fig. 8. Effect of rotation by angles (b) 11°, (c) 22° and (d) 45°.

3.4. Data Reduction Efficiency

The above neural network model using competitive learning creates an adaptive vector quantisation [4] of the input set. Each weight vector tends to the centroid of the respective regions S_i. Within each region, all pixels have the same weight vector as their nearest one. In general, the output weight vectors give the prototype or examplar vectors from the appropriate classes (regions), and these can be an encoded version of the input, in less storage space. In the present situation, the set of weight vectors along with their interconnections, or the graph (planar straight line graph) with the weight vectors as its nodes and the interconnections as its edges, represents the skeletal shape of the input binary pattern. The graph requires much less space than the original input set, and hence a considerable data reduction is achieved.

One of the basic purposes of skeletonisation is to reduce the storage space required to store the image data without losing the essential structural information. The proposed method can achieve more data reduction compared to most of the existing skeletonisation algorithms. It can be seen that the fewer processors in the network, the greater the data reduction. By choosing larger values of δ_1 and δ_2, i.e. by making the average distance between neighbouring processors larger, we can make higher data reduction. But this might worsen the accuracy of medial axis representation. A proper choice of δ_1 and δ_2 (as mentioned earlier) can balance this trade-off.

3.5. Extendibility to Dot Patterns and Grey-Level Images

For a dot pattern, unlike in the case of a binary image, a skeleton is not be properly defined. But our biological visual system can still extract the *perceptual skeleton* from a dot pattern. For example, a dot pattern having a definite shape (say, 'A'-like) can be recognised by the human brain almost as easily as a binary image having the same shape. The conventional thinning algorithms that extract skeletons from binary images do not work for dot patterns. In contrast, as seen in Section 3.1.2, the proposed neural algorithm, with a minor modification, can be used to extract a perceptual skeleton of a dot pattern [15].

Conventional binary image thinning algorithms work only on binary images. They cannot be extended to grey-level images. On the other hand, the neural technique discussed here can also be extended for implementation to grey-level images. A grey-level image may concern either the whole image or a subset of it. Here the latter case is assumed. We consider, as in Arcelli and Ramella [16], grey-level images where the area of interest (i.e. the object portion) can be interpreted as constituting a multi-valued foreground emerging from a single-valued background. The extension can be done in the following way.

Suppose for a grey-level pattern, g_{rs} is the grey value of the pixel at the rth row and sth column. Then the update rule replacing Eq. (2) will be

$$W_p(t+1) = W_p(t) + \alpha(t)[P_j - W_p(t)]g_{rs}, \quad \text{for } \pi_p \in N_k \tag{12}$$

Multiplication by g_{rs} means that the amount of update will be more for pixels with high grey values and less for pixels with low grey values. The grey-level extensions for arc and tree-like patterns are thus straightforward. But for loop patterns, the extension is not trivial because the definition of adjacency is not well defined in that case. If criteria for the adjacency are available, then the extension is also possible for loop patterns. In some situations, the loop joining criteria used for dot patterns may work. Alternatively, one can use the adjacency as defined for binary images after local thresholding (locality being defined by the regions S_i). The algorithm has been tested on several grey-level images, and Fig. 9 shows the result from a chromosome image.

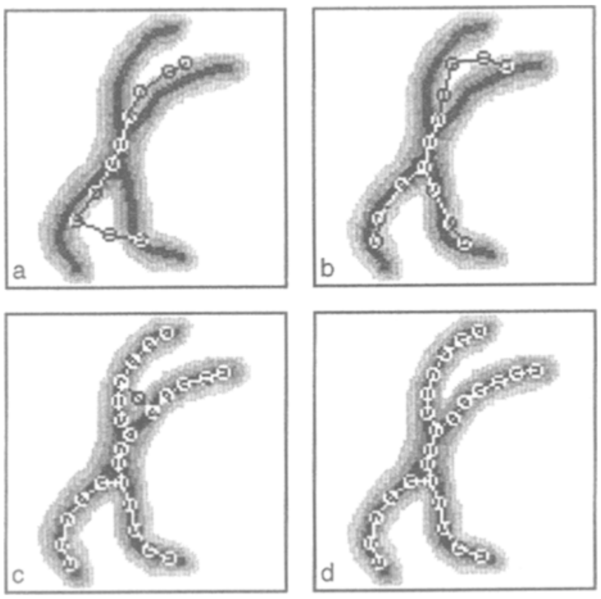

Fig. 9. Output skeleton of a grey-level chromosome image with intermediate stages of skeletonisation.

4. Conclusions

Performance analysis of a neural network-based skeletonisation technique for a binary object is the main motivation here. The technique uses a modified version of Kohonen's self-organising model. This neural technique is compared with a number of conventional thinning techniques qualitatively as well as quantitatively. The comparative studies, carried out here, demonstrate that the neural technique has some advantages over the existing conventional thinning techniques. Most of the existing conventional algorithms are not robust to noise; some of them are robust to boundary noise only. The neural algorithm is highly robust to both boundary and interior noise, in the sense that it can produce satisfactory skeletons even under very low SNR (close to 1) or a very high amount of noise. Noise is unavoidable in real-life applications. The neural technique is useful particularly in a highly noisy environment.

Another advantage of the neural algorithm is that it is invariant under rotation by arbitrary angles. Most of the existing algorithms are invariant under rotation of 90° only. The data reduction efficiency of the neural algorithm is higher than that of the other algorithms. Moreover, the neural technique is found to be extendible to a dot pattern or to a grey-level image. Thus the neural technique provides us with a unified approach to skeletonisation.

References

1. Smith RW. Computer processing of line images: a survey. Pattern Recognition 1987; 20: 7–15
2. Lam L, Lee SW, Suen CY. Thinning methodologies – a comprehensive survey. IEEE Trans. PAMI 1992; 14: 869–885
3. Datta A, Pal T, Parui SK. A modified self-organizing neural net for shape extraction. Neurocomputing 1997; 14: 3–14
4. Kohonen T. Self-Organization and Associative Memory, Springer-Verlag, 1989
5. Sabourin M, Mitiche A. Modeling and classification of shape using a Kohonen associative memory with

selective multiresolution. Neural Networks 1993; 6: 275–283
6. Fritzke B. Let it grow – self-organizing feature maps with problem dependent cell structure. In: Kohonen T et al. (eds), Artificial Neural Networks, vol. 1, North-Holland, 1991
7. Choi D, Park S. Self-creating and organizing neural networks. IEEE Neural Networks 1994; 5: 561–575
8. Kangas JA, Kohonen T, Laaksonen J. Variants of self-organizing maps. IEEE Trans Neural Networks 1990; 1: 93–99
9. Datta A, Parui SK. A robust parallel thinning algorithm for binary images. Pattern Recognition 1994; 27: 1181–1192
10. Jang BK, Chin RT. One-pass parallel thinning: analysis, properties and quantitative evaluation. IEEE Trans PAMI 1992; 14: 1129–1140
11. Arcelli C, Cordella L, Levialdi S. Parallel thinning of binary pictures. Electron Lett 1975; 11: 148–149
12. Tamura H. A comparison of line thinning algorithms from digital geometry viewpoint. Proc 4th Int Joint Conf Pattern Recog, Kyoto, Japan, 1978; 715–719
13. Hilditch CJ. Comparison of thinning algorithms on a parallel processor. Image Vision Comput 1983; 1: 115–132
14. Jang B, Chin RT. Analysis of thinning algorithms using mathematical morphology. IEEE Trans PAMI 1990; 12: 541–551
15. Datta A, Parui SK. Skeletons from dot patterns: a neural network approach. Pattern Recognition Letters 1997; 18: 335–342
16. Arcelli C, Ramella G. Finding grey-skeletons by iterated pixel removal. Image and Vision Comput 1995; 13: 159–167