1988

Skeletonization, thinning and thickening algorithms and their applications to Arabic characters.

Mohamed. Tellache
University of Windsor

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, tests publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30.

Canada

SKELETONIZATION , THINNING AND THICKENING ALGORITHMS AND THEIR APPLICATIONS TO ARABIC CHARACTERS

by

MOHAMED TELLACHE

Submitted to the
Faculty of Graduate Studies and Research
through the Department of
Electrical Engineering in Partial Fulfillment
of the requirements for the Degree
of Master of Applied Science at
the University of Windsor

Windsor, Ontario, Canada
1988

ISBN 0-315-48172-2

DEDICATION

TO MY PARENTS

ABSTRACT

This thesis discusses the development of thinning and thickening algorithms for the purpose of thinning or thickening ' camera captured ' Arabic fonts.

Skeletonization is a widely used preprocessing technique in 'linearwise' image analysis, for example, analysis of alphanumeric printed or handprinted characters, Chromosomes, etc. One of the methods for obtaining the 'skeleton' or a 'medial line' of a binary picture, consists in erasing points of the boundary of the object without modifying the topological properties of the picture until it consists of thin lines.

Various existing thinning algorithms are presented and compared to the algorithms presented in the thesis.

Two new parallel thinning algorithms are developed. One is based on local operations to detect edge-points, end-points and break-points. The other is a matching algorithm in which a set of eight templates and two images (the current image and the working image) are used in the processing. Both algorithms maintain the connectivity, conserve the shape of the original image and do not leave extraneous pixels within a 3-by-3 window to produce a pixel thin skeleton from a connected thick image. Also a thickening algorithm is presented.

Experimental results show that these methods are very effective for thinning and thickening of Arabic fonts.

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to my supervisor , Dr.M.A.Sid-ahmed for his valuable suggestions , guidance, support, and helpful remarks throughout the course of this work.

I am also most grateful to Dr. J. J. Soltis, Dr. H. K. Kwan, and Dr. V. M. Huynh for their helpful contributions and constructive suggestions.

I wish to thank all my fellow graduate students and my colleques, A.NAMANE , M.M.MAMERI , F.EL-KHALI , S.O.BELKASIM. I would like to thank all my friends from Electrical (Power) and Civil Engineering departments.

Last but not least, I extend my sincerest thanks to all my beloved parents, my brother Youcef and all my familly in Algeria.

TABLE OF CONTENTS

FIGURE CAPTIONS

Chapter I

INTRODUCTION

Thinning algorithms have been studied widely in picture processing and pattern recognition because they offer a way of simplifying pictorial forms.

Our goal is to perform skeletonization , thinning and thickening algorithms on Arabic characters.

Thinning algorithms are used for different purposes such as:

1. Automatic analysis of Chromosome spreads
2. Quantitative measurement of soil cracking patterns.
3. Feature extraction , printed character recognition and image representation.
4. Type-setting characters

Thinning is the most common name for the process of reducing the width of a line-like object from many pixels wide to minimal width.

Other terms commonly used to describe the determination of a single pixel are " skeletonization " , " medial axis transformation " and " symmetric axis transformation ".

One advantage of skeletonization is to reduce the memory space required for storing the essential structural

information present in the patterns. Secondly, it also simplifies data structures required in processing the pattern.

There are several possible definitions of a skeleton :

1. Montanari [5] defines a skeleton as being the result of propagating wavefronts from the inside of the edge of the figure. The skeleton is then the locus of the intersections of wavefronts from opposite sides. This is roughly equivalent to the scheme adopted by most authors, where the " outside " pixels are removed iteratively from the image until no more can be removed without disconnecting one arc from another or shortening a line.

2. Rosenfeld [6-7] takes the definition that a skeleton is formed from the centers of maximal discs placed within the object. Using this definition the skeleton can be used to reconstruct the original object accurately by retaining the radii of the maximal discs and simply re-drawing them. This definition however, does not guarantee that the skeleton will be connected (i.e. one guarantee that the skeleton will be converted to several black with white spaces in between).

3. Davies and Plummer [8] define a skeleton as being the result of (2) but with sufficient additional points such that the resulting skeleton is connected. This is the most useful of the definitions by combining a strict definition with connectivity.

4. Pratt [3] defines shrinking and thinning as operations that seek to reduce a connected region of pixels of a given properly set to a smaller size. Shrinking reduces the region to a single pixel, while thinning reduces a region to minimal cross-sectionnal width. Skeletonizing is a related operation that transforms a region to a stick figure representation.

In general thinning algorithms are used to reduce an elongated object to one element thick.

Most existing thinning algorithms take the following processing route, starting with the analog output of the scanner device.

a-Thresholding :The scanner hardware usually converts the image to 0 (white) or 1 (black) before passing it to the computer (see APPENDIX - A -).

b- Thinning :An iterative edge erosion process removes outside pixels until only the skeleton remains.

1.1 OVERVIEW ON DIFFERENT TECHNIQUES:

In this section, a review of various thinning algorithms described in the literature is presented.

A thinning algorithm is considered useful for a particular application if :

1. It preserves the shape of the original image.
2. It preserves the connectivity
3. It does not leave extraneous pixels (branches).

Several authors [9-11] have studied the problem of obtaining sets of thickness invariant measurements which involve the determination of constant thickness "medial line" of each object and considering for recognition purposes only the features of the line , unfortunately these measurements are invariant only with respect to a uniform change of thickness in the entire picture.

Some of the work in the literature [12-15] is concentrated on two fundamental approaches. The medial axis transform (MAT) and the distance transform (DT). The strength of the (MAT) is its ability to generate image skeletons that are always connected . The DT on the other hand produces skeletons that may have gaps but can be implemented to run faster on a conventional general purpose computer.

Alcorn and Hoggar [16] discussed the work of Saraga and Woollons [17] , Dinneen [18] , Unger [19] , Sherman [20] and Deutsch [13] who all used various sized window operators

which are passed iteratively over the whole image to decide whether a point is to be deleted . The paper concentrates on the follow-up to the work of Dinneen . A 3-by-7 window is passed iteratively over the whole (binary) image. The number of black pixels in the window is counted and if this figure is below a fixed threshold value, then the central pixel is set to white. If the count is above a different value, the central pixel is set to black. According to the values of these thresholds the result is that either small gaps in the black area are filled in, or thinning is performed by eroding the edge from the black region.

The shape of the window has the unfortunate effect of producing skeletons with thin vertical lines and thick horizontal lines , so the final " skeleton " is not consistently of unit width.

Beum [21] started with a binary image and uses an iterative technique with 2 passes per iteration. The first pass marks all edge points as being eligible for deletion. The second pass then deletes all marked points which are not critical to the skeleton i.e. points which would not cause a discontinuity or an end of a line to be shortened . The two passes are then used iteratively until no more points are deleted . A final " clean-up " pass is used to reduce the image to a stick line of one pixel thin. In the paper it is suggested that a number of iterations be fixed to say 3. The clean-up pass is then relied on to complete the thinning.

The chief difference between this algorithm and [16] is in the organization of the 2 passes. Hilditch has the second pass delete all marked points. Beum performs the majority of the checks to prevent deletion of break-points and excessive erosion on the second pass, the checks being only done on edge points marked on the first pass.

Udupa and Murthy [22] suggested a method for converting a binary image directly to a set of points which form a line segment approximation of the skeleton. The algorithm uses an iteration where a set of window operators over the image to detect " turning points " and " end-points " in the image. Unfortunately the algorithm yields a skeleton of a constant width over at least 6 pixels.

R.Stefanelli and A.Rosenfeld [23] presented a basic scheme of thinning which consists of a sequence of four parallel steps thinning the objects in four cardinal directions (north , south , west , east) successively. The disadvantage of this method is the existence of branches in the final skeleton.

A.Rosenfeld and L.S.David [24] described a simple algorithm for thinning object down to skeletons and a method of detecting and pruning "noisy" branches of a skeleton, with the aid of the distance transform of the original object. The main problem of this algorithm is that the final skeleton may not be connected.

Tamura [25] described a thinning algorithm that aims at two important properties. The first is of a topological nature and consists of preserving the order of connectivity of both object and background. The second is the definition of end points by which the skeleton begins to emerge from the objects.

C.C.Lee [26] described a modified distance transform (MDT), which combines the original distance transform (DT) with a new set of selected rules to be defined, and an appropriate linking algorithm to produce a connected skeleton in computing times that are significantly shorter than the medial axis transform (MAT) implementation for a large image array. It should be noted that by using this method, the skeleton of one pixel width cannot be obtained.

Chung Chang Lee [27] presented a sequential thinning algorithm which maintains a connectivity within 3-by-3 window to produce a pixel thin skeleton from a connected thick skeleton. The process is based on local neighborhood logic skeletonizing operations for fast processing, and consists of :

1. Sequential application of a modified distance transform ;
2. linking algorithm ; and
3. thinning algorithm

The algorithm is excellent concerning the processing time and preserves very well the shape of the original image

but it has sometimes the problem of discontinuity when it is applied to Arabic characters .

For Arcelli [30] a skeleton can be obtained using a matching algorithm in which a set of eight templates is used. The problem of this method is the existence of branches in the final skeleton when only one image is used in the processing.

For Carlo Arcelli and G.S.Di-Baja [29] the thinning is obtained using an algorithm which transforms a digital figure into a set of simple digital arcs and curves, by employing local sequential operations . The procedure considers the removal of suitable contour elements of the given figure and it is repeated on every current set until the final skeleton is obtained.

A.Bel-Lal and Montoto [32] presented a thinning algorithm which determines the "medial line's " elements of an elongated object on a digital image and , at the same time a value for those elements equal to the "width" of the object . The procedure deletes boundary points of the object in an iterative process , a way which is widely used in the bibliography to find only the position of the medial line's element. However using this method the skeleton contains branches originating in correspondence to all the deletable convex arcs of the contour.

C.Arcelli , L.P.Cordella and S.Levialdi [33] described a parallel procedure applied to a connected image , the

procedure involves a step by step propagation of the background over the image. At every step, contour elements either belonging to the significant convex regions of the current image or being local maxima of the original image are selected as skeleton elements.

For the basic algorithm of Pavlidis [39] ; a dark point P is deleted from the pattern if it satisfies all of the following conditions: 1) it is an edge-point; i.e. it has at least one white four-neighbors; 2) it is not an end-point; and 3) its neighborhood does not match any of three predefined 3-by-3 rotating windows (Pavlidis calls these " multiple points " and " tentative multiple points " tests). Effectively , these tests ensure connectedness of the skeleton and prevent excessive erosion .

Hilditch [12] presented a thinning algorithm based on the following criteria : a dark point P is deleted if it satisfies all of the following conditions : 1) it is an edge-point; 2) it is not an end-point; 3) it has at least one dark eight-neighbors not considered as deletable; 4) it is not a break-point; i.e., its neighborhood satisfies the " crossing number " criteria (see N.J.Naccache algorithm [35]); 4) its deletion does not cause excessive erosion.

R.Stefanelli and A.Rosenfeld [23] presented a "simplified version" of Hilditch's approach [12] described below. In this algorithm, a dark point P is deleted if it satisfies the following conditions: 1) it is an edge-point;

2) it is not an end-point; 3) its deletion does not cause excessive erosion; 4) it satisfies the break-point test; i.e. the number of dark to white transitions in the neighborhood of P is equal to 1. The main problem of using this method is that the shape of the original image is not always preserved and also sometimes the presence of a few extraneous pixels (branches) either by flagging the pixels in each pass and deleting the flagged pixels after the last iteration, or deleting a pixel each time the four conditions are satisfied.

Nabil Jean Naccache and Rajjan Shinghal [35] presented an algorithm based on local operations to detect edge-points, end-points and to prevent excessive erosion; also compare their results with its so called "Simplified version of Hilditch" presented by Stafenalli and Rosenfeld [23]. The algorithm consists of six conditions. One of the main disadvantages is the existence of branches, the other is that the final skeleton is not connected.

For A.Rosenfeld [42] the natural concepts of simple connectedness are defined for subsets of a digital picture . It is shown that every simple connected object (with more than one element) in such a picture has elements which can be deleted without destroying its simple connectedness.

Rosenfeld and Pfaltz [40] defined a distance transform of a binary image. This replaces each pixel by a number

indicating the minimum distance from that pixel to the white point. The distance between two points is defined by the number of pixels in the shortest 4-connected chain between the points. This transformation is calculated by evaluating a function sequentially in a raster scan over the image, followed by a second function in a reverse scan. Once the distance function is calculated, a local maximum operation is used to find the skeleton. It is shown that this is the smallest set of points needed to reconstruct the image exactly. The main problem with this algorithm is that the skeleton may not be connected. However, the time required is of the order of the number of pixels in the image, making this form of thinning faster than iterative algorithms.

It should be noted that the strokes thinned by hardware or software are accompanied by different kinds of distortion. Different algorithms produce different degrees of distortion [2,3,22,13].

1.2 OBJECTIVE OF THE RESEARCH

The objective of this research is to investigate the utility of thinning and thickening algorithms to Arabic characters, and to develop new algorithms that will maintain the connectivity, preserve shape of the original image and avoid such problems as extraneous pixels (branches).

1.3 THESIS ORGANIZATION :

The thesis is divided into three parts

Some basic definitions, illustrations and procedures are presented in chapter II.

In chapter III various parallel and sequential thinning algorithms are presented and studied in detail. We have shown the problems of each algorithm when applied to Arabic characters.

In chapter IV two new parallel thinning algorithms are developed to investigate the problems discussed in chapter III, and applied to Arabic characters. Also a thickening algorithm is presented

Chapter V presents the conclusions that can be obtained from the research work presented in this thesis.

Finally some other examples are given in Chapter VI. In this Chapter all thinning algorithms presented in Chapter III and IV are applied to hand-written and printed English fonts and some numbers.

Chapter II

ILLUSTRATION , DEFINITIONS AND PROCEDURES

2.1 INTRODUCTION :

Usually different researchers have described various formal tests to prevent excessive erosion and to detect edge-points, end-points and break-points. Before going on to describe these procedures it would be useful to give brief illustrations and definitions for thinning approaches.

2.2 GENERAL ILLUSTRATION :

Figures 2.1,2.2,2.3 and 2.4 illustrate the operation of a simple thinning and skeletonization for regularly shaped regions and irregularly shaped regions which consists of removing border points that are not end-points and do not result in a disconnected region under the definition of eight-connectivity.

a- Regularly shaped region :

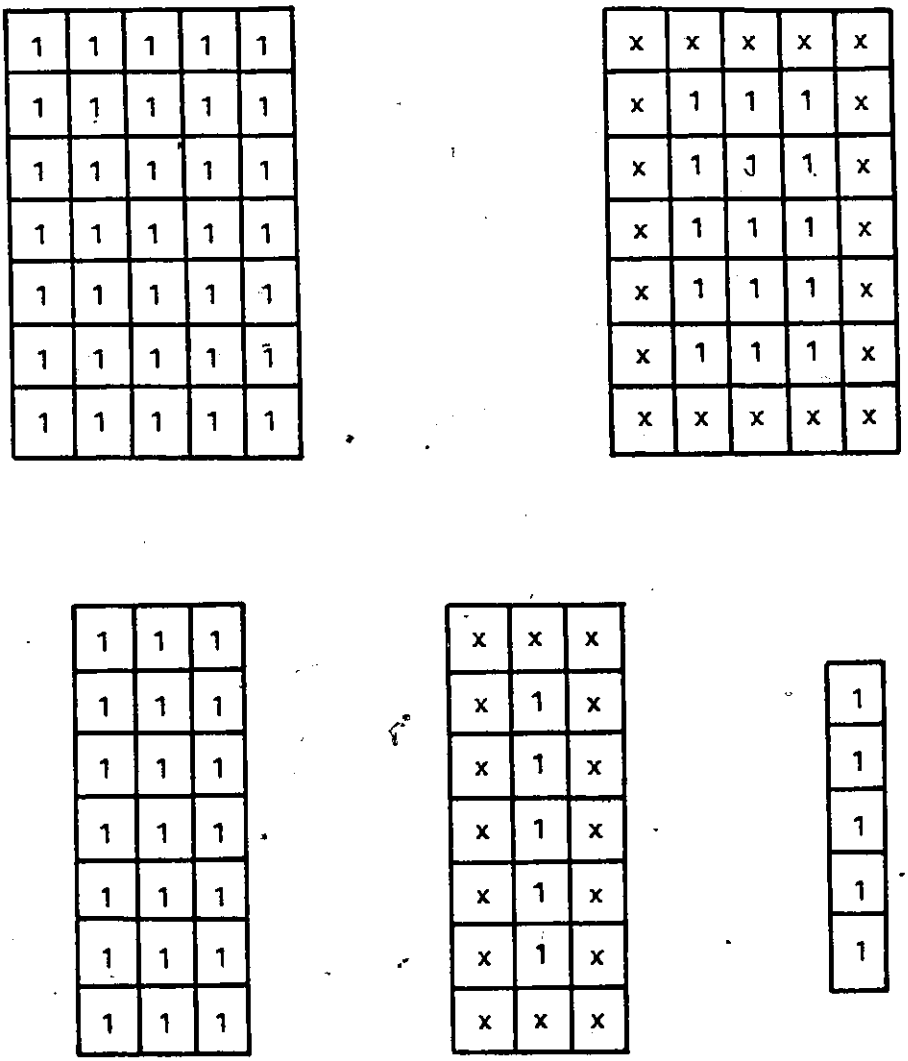


Figure_2.1: Case of rectangle : The value 1 corresponds to object pixel and x corresponds to deletable pixels.

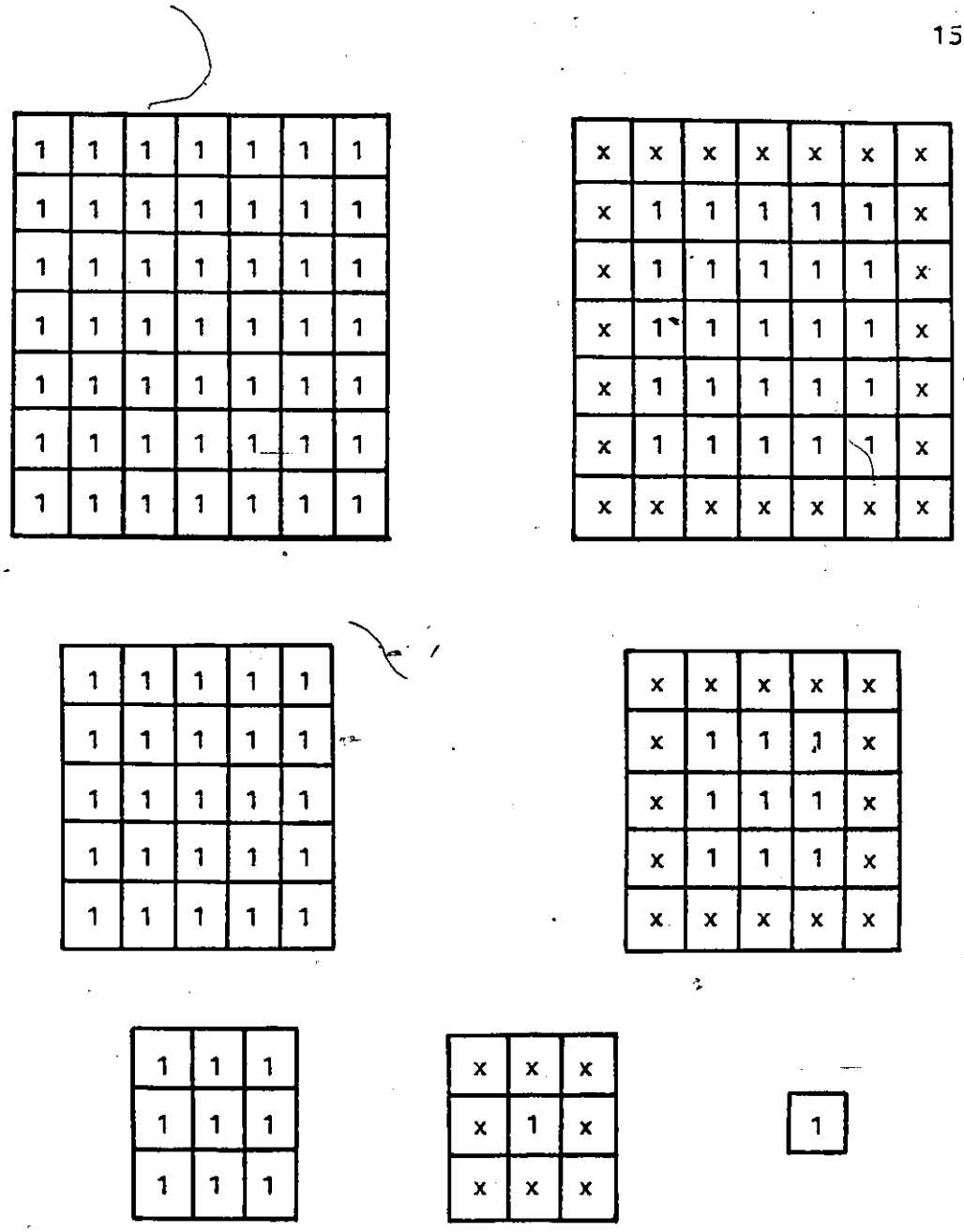


Figure 2.2: Case of square

b- Irregularly shaped region

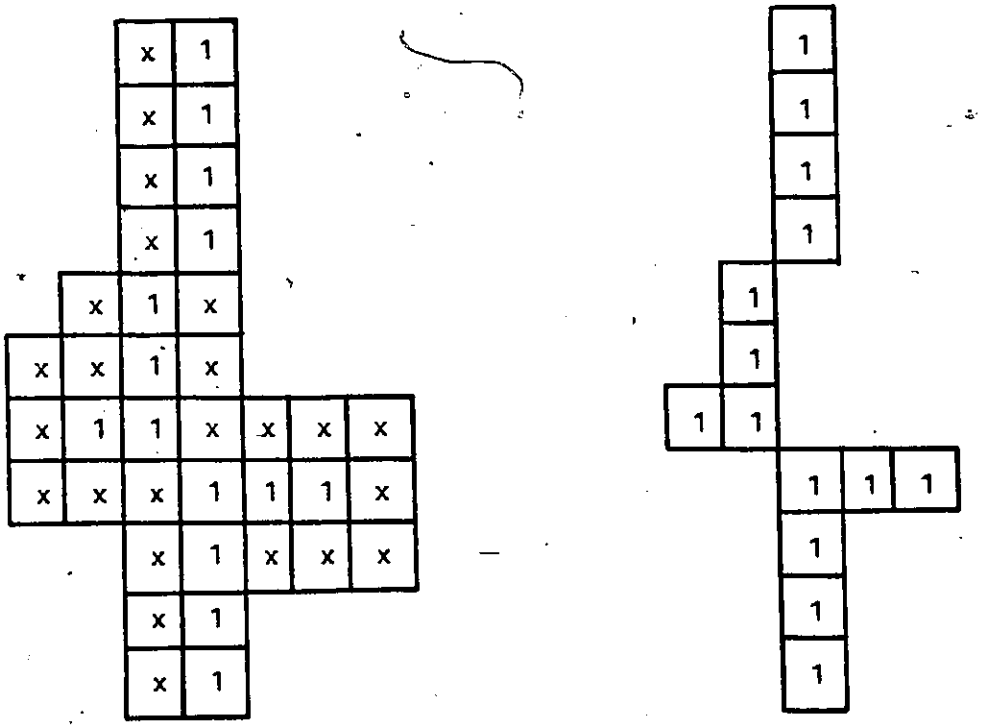


Figure 2.3: Vertical region thinning

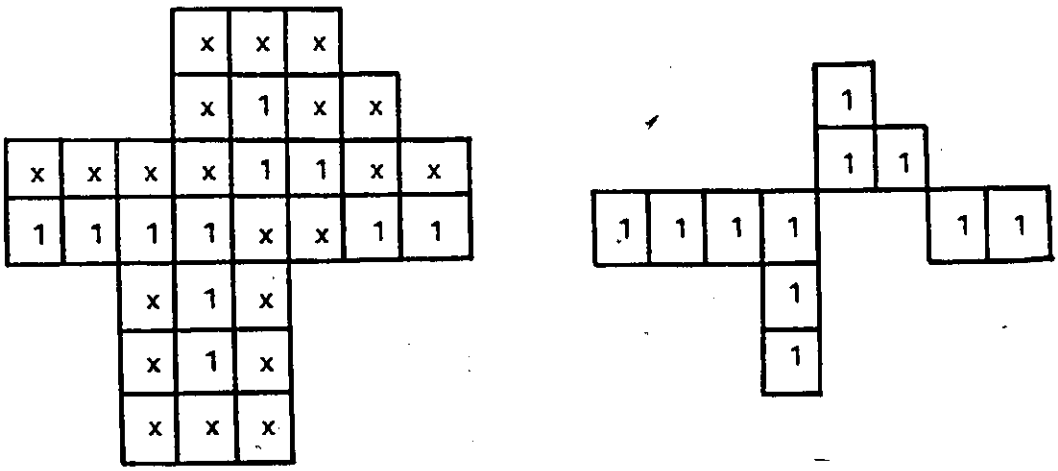


Figure 2.4: Horizontal region thinning

2.3 = DEFINITIONS

It is possible to define thinning in a mathematically rigourous way on the continuous plane as follows.

2.3.1 _DEFINITION(1)_[2]

Let R be a plane set, B its boundary and P a point in R. A nearest neighbor of P on B is a point M such that there is no other point in B whose distance from P is less than the distance PM.

If P has more than one nearest neighbor, then P is said to be a skeletal point in R. The union of all skeletal points is called the skeleton or medial axis of R.

This definition implies that skeletal points are centers of circles contained entirely within R with the property that there is no other circle with the same center and greater radius contained in R.

Figure(2.5) shows some examples of skeletons with some of their major features.

One can see that they are very sensitive to noise , since a small disturbance of the boundary not only causes a disturbance in one branch but also causes the creation of new branches

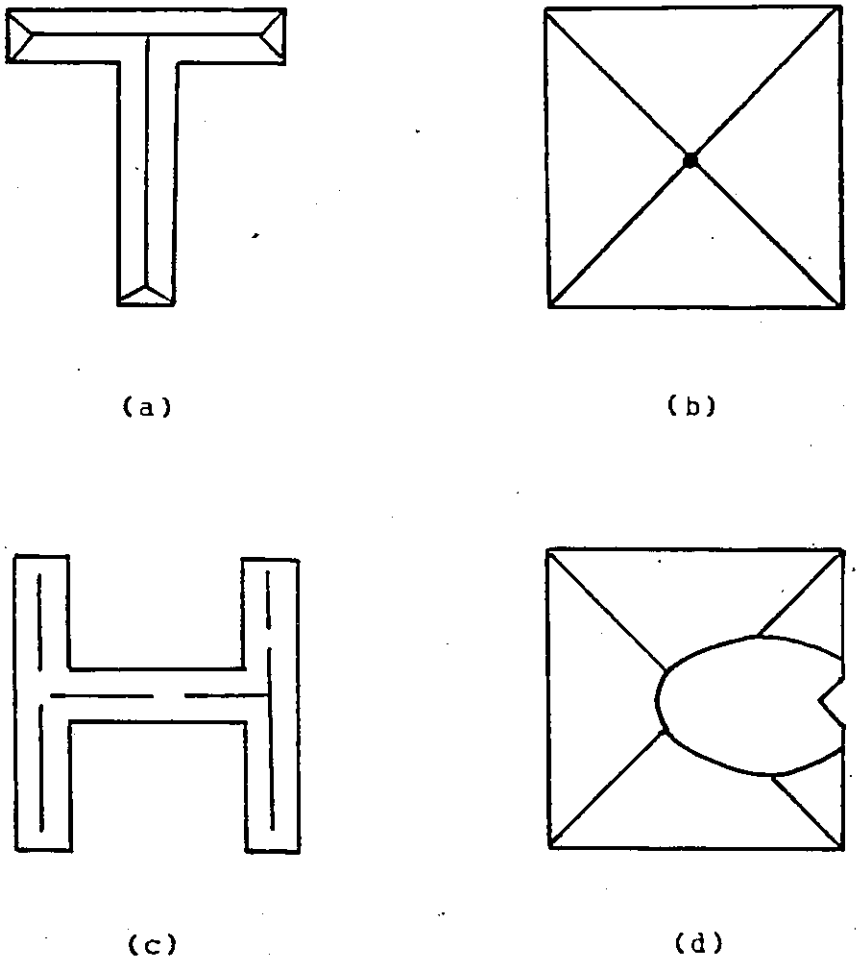


Figure_2.5: Examples of skeletons

(a). The linear structure of the character shown in this figure corresponds closely to that of the medial axis.

(b). There is no simple correspondence between skeleton branches and the square, because if this is the case we should get a point as a skeleton.

(c). The medial axis of the character shown in this Figure is disconnected.

(d). This figure shows that a small amount of noise alters the form of the skeleton .

Note that medial axis means the line which lies in the middle of the thick object , in other word the skeleton of the thick image.

Another observation that one can make from Fig.(2.5) is that for objects that are thin to start with, the skeletons provide substantial information about their shape. This is not the case with a thick object , as shown in Fig.(2.5b).

2.3.2 DEFINITION(2)

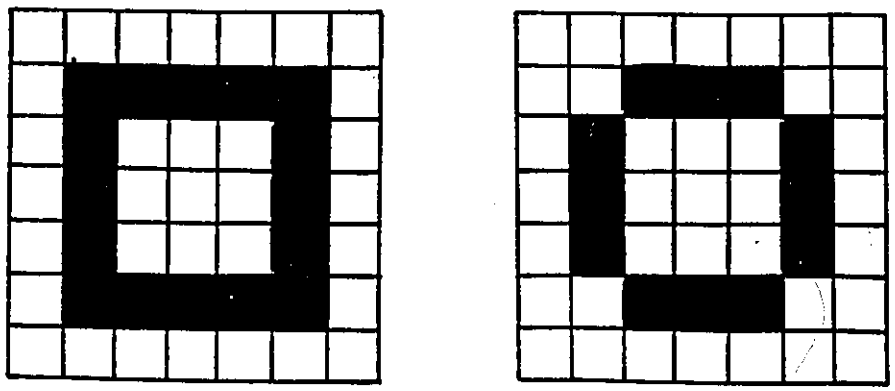
The skeleton of a set of pixels R is a set found as follows.

First the skeletal pixels and the contour pixels of R are determined. Then all the contour pixels that are not skeletal are removed and the set thus found replaces R. The process is repeated until a set consisting of only skeletal pixels is left.

The important part is the assumption that one can decide for each set that certain pixels are skeletal, and keep them, and the other pixels are definitely not skeletal, so one can discard them.

2.3.3 DEFINITION(3) : CONNECTIVITY [3].

A fundamental step in the formation of a symbolic description of a picture from an array of pixels or a collection of primitive features is to specify the geometrical relationship or connectivity of pixels . In the binary picture of Fig.(2.6a) the ring of black pixels, by all reasonable definitions of connectivity , divides the picture into three pixel regions : the white pixels exterior to the ring , the white pixels interior to the ring , and the black pixels of the ring itself. The pixels within each region are said to be connected to one another. This concept of connectivity is easily understood in Figure(2.6a) but ambiguity arises when considering Figure(2.6b).



(a) (b)

Figure 2.6: Illustration of connectivity : a- Ring figure ; b- Ambiguous figure

Neighborhood operations lend themselves naturally to the study of connectivity, since it is defined in terms of neighbors. If the operations are applied in parallel, it is not easy to distinguish among connected components, since the parallel operations treat every point of the given set identically (see section 2.4.4). Using the sequentially applied operations, however, one can track each connected region, assigning a value to each point of it as the tracking proceeds.

2.4 PROCEDURES

2.4.1 SKELETONIZATION CRITERIA

In image processing skeletonization of binary patterns consists of successive deletions of dark points (i.e changing them to white points) along the edges of the pattern until the pattern is thinned to a line drawing.

Ideally , the original pattern should be thinned to its medial axis.

It means the procedure deletes boundary points of the object in an iterative process to find only the position of the medial line's element. Simultaneously calculation of position and width was found in using two arrays, one to place the position and the other to record the width.

Most skeletonization algorithms consist of iteratively executing many passes over the pattern where in each pass a few dark points are deleted. In any pass, a dark point to

be deleted from the pattern must satisfy the following intuitive criteria.

a- It is an edge-point (i.e it lies along the edges of the pattern

b- It is not an end-points (i.e it is not a point which lies on the extremities of the stroke).

c- It is not a break-point (i.e it is not a point the deletion of which would break the connectedness of the pattern).

d- Its deletion must not cause excessive erosion (e.g an open ended stroke should not be iteratively deleted).

2.4.2 LOCAL OPERATIONS

By an operation on a digitized picture it is meant a function which transforms a given m*n picture matrix into another one. A general function of this type has m*n numericals arguments (one for each position in the matrix), and is correspondingly difficult to handle computationally. For practical purposes, it is desirable to work with operations on digitized picture which can be defined in terms of functions having considerably fewer arguments.

By a local operation or neighborhood operation on a picture it is meant a function which defines a value for each element in the transformed picture in terms of the values of the corresponding element and a small set of

neighbors in the given picture. For example , such operations can be defined using a neighborhood which consists of the given element and its eight immediate neighbors, an operation of this type has only nine arguments and is of the form :

$$a^{*}_{i,j} = f(a_{i-1,j-1}, a_{i-1,j}, a_{i-1,j+1}, a_{i,j-1}, a_{i,j}, a_{i,j+1}, a_{i+1,j-1}, a_{i+1,j}, a_{i+1,j+1})$$

When processing a picture using local operations, the purpose is to share the results of intermediate processing steps as auxiliary pictures.

The processing for thinning algorithms can be serial (sequential) or parallel.

2.4.3 SEQUENTIAL PROCESSING

Suppose that a local operation is applied to the points of a digitized picture in some definite sequence. For simplicity, suppose that the points are processed row by row beginning at the upper left.

In serial (or sequential) processing, one point is processed at any one time ; the result of processing a point at the nth+1 iteration depends on a set of points obtained at the nth iteration . The processing sequence can be preassigned (for example , as a television type scanning of the visual field) or computed (the next point to be processed depends on the results of the previous processing

; this is the case, for example in all contour following algorithms). In other words, the new value rather than the original value is used in processing any succeeding points which has it as a neighbor.

If g(i,j) and g'(i,j) denote respectively , the values of the input and output images of an elementary step in point (i,j) then

$$g(i,j) = F(\, g'(i-1,j-1), g'(i-1,j), g'(i-1,j+1), g'(i,j-1), g(i,j), g(i,j+1), g(i+1,j-1), g(i+1,j), g(i+1,j+1))$$

Note that the points (i-1,j-1) , (i-1,j) , (i-1,j+1) and (i,j-1) have already been processed , while the remaining points have not yet been processed.

2.4.4 PARALLEL PROCESSING

The wide variety of picture transformations which can be performed using local operations applied in parallel has given rise to the widespread belief that this approach is optimum for local picture processing in general.

In parallel processing the value given to a point at the nth iteration depends on the values given to the point and its eight neighbors at the (n-1)th iteration. Thus all the points of the figure can be processed simultaneously.

In a parallel algorithm , the elementary function F is applied at once to all points in the neighborhood .

$$g'(i,j) = F(\, g(i-1,j-1), g(i-1,j), g(i-1,j+1), g(i,j-1), g(i,j), g(i,j+1), g(i+1,j-1), g(i+1,j), g(i+1,j+1)).$$

The arguments (q(i-1,j-1),.....,q(i+1,j+1)) are always the original picture matrix values , the new values q'(i,j) = F(q(i-1,j-1),.....,q(i+1,j+1)) are stored but are not used until the operation has been performed for every (i,j) , when they then become arguments for the next operation (if any).

Note that sequential processing can be more efficient than parallel processing when it is implemented on a general computer. On the other hand parallel processing is advantageous if suitable special purpose computing structures can be used , e.g cellular networks.

It was noted in [40] that any picture transformation that can be accomplished by a series of parallel local operations can also be accomplished by a series of sequential local operations , and conversely the same is true.

Chapter III

VARIOUS THINNING ALGORITHMS WITH APPLICATION TO ARABIC CHARACTERS

3.1 INTRODUCTION

The key step of many pattern recognition tasks is a thinning algorithm. One of the methods of obtaining a skeleton or a medial axis of a binary picture, consists of erasing points of the boundary of the object without modifying the topological properties of the picture until it consists of thin lines.

There are many ways to perform thinning transformation when applied to a binary picture such as:

1. MATCHING ALGORITHMS USING TEMPLATES
2. USING LOCAL OPERATIONS
3. DISTANCE TRANSFORM

Before going on to describe some thinning algorithms, it would be useful to those unfamiliar to the field to define some common terminology.

Let S be a subset of the digital picture, we sometimes refer to the points of S as 1's and to the points not in S as 0's.

If P is a point of S , and the neighborhood of P is as denoted in Fig.(3.1)

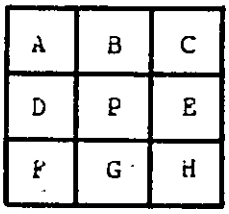


Figure_3.1: Neighborhood of P

Then we call P a north border point if B = 0 ; an east border point if E = 0 ; a west border point if D = 0 , a south border point if G = 0, a north east corner point if B = E = 0, a east south corner point if E = G = 0, a south west corner point if G = D = 0, and a west north corner point if D = B = 0.

The points B , D , E and G are called 4-neighbors of P (called also the 1st order neighbors of P or orthodiagonal neighbors) as shown in Fig.(3.2).

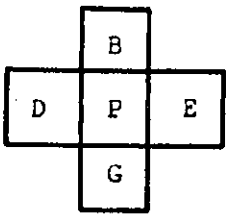


Figure_3.2: The four-neighbors of P

while these points together with A , C , F and H are called the 8-neighbors of P (called also the 2nd order neighbors of P).

By the definition of four-connectivity, pixel P and pixel E are connected if both belong to property set S . Similarly four-connectivity can be established between pixel P and pixels B , D and G , which all share an extended boundary , provided that both members of the pair belong to the same property set.

Eight-connectivity permits pixel P and one of its diagonal neighbors , with a common point boundary , for example pixel P and pixel C to be connected if they both belong to property set S .

Note that pixel P belongs to the property set S which means that pixel P is a point of S with 1 valued pixel.

Mathematically S is called 4-connected if for any two points P and Q of S there exists a sequence of points

$$P = P_0 , P_1 , P_2 , \ldots\ldots\ldots , P_n = Q$$

of S such that P_i is a 4-neighbor of P_{i-1} , $1 \leq i \leq n$

S is called 8-connected if for any two points P and Q of S there exists a sequence of points

$$P = P_0 , P_1 , P_2 , \ldots\ldots\ldots , P_n = Q$$

of S such that P_i is an 8-neighbor of P_{i-1} , $1 \leq i \leq n$

We call P a 4-end point if exactly one of its 4-neighbors is one (1), and an 8-endpoint if exactly one of its 8-neighbors is 1.

For example , in

0	0	0
0	P	1
1	0	1

P is 4-end point (but not 8-end point).

We call P 4-isolated if none of its 4-neighbors is 1 , and an 8-isolated if none of its 8-neighbors is 1.

We call a border point P 4-simple if changing it from 1 to 0 does not change the 4-connectedness of the 1's in its neighborhood ; and a 8-simple if changing it from 1 to 0 does not change the 8-connectedness of the 1's in its neighborhood.

For example

0	0	0
0	P	0
1	0	1

P is 4-simple but not 8 simple ; in

0	0	0
1	P	1
0	0	0

it is neither ; in

0	0	0
0	P	1
0	0	1

it is both.

Readily , a 4-endpoint is always 4-simple and an 8-endpoint is 8-simple, and the same is true for isolated points.

A pixel P is a break point if it has a value of 1 and its 3-by-3 window matches any of the following six window patterns denoted by S_1, S_2, S_3, S_4, S_5 and S_6 as shown in Fig.(3.3).

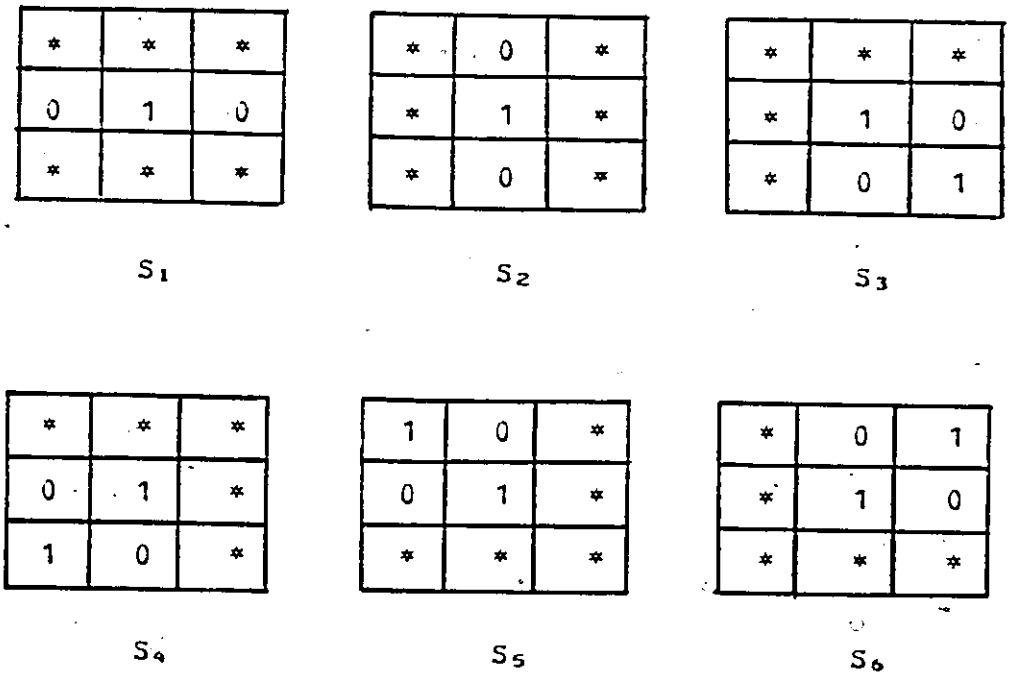


Figure 3.3: The six windows S_1, S_2, S_3, S_4, S_5 and S_6, where x represents don't care conditions (i.e can have either the values 0 or 1)

3.2 ALGORITHM #1 (MATCHING ALGORITHM) [30-31]

3.2.1 INTRODUCTION

The concept of template matching has found wide acceptance in many applications because of its simplicity.

In terms of digital image, a template (also called a mask or window) is an array designed to detect some invariant regional property.

One of the most fundamental means of object detection within an image field is by template matching, in which a replica of an object of interest is compared to all unknown objects in the image field. If the template match between an unknown object and the template is sufficiently close, the unknown object is labeled as template object.

3.2.2 ARCELLI'S ALGORITHM [30]

The matching algorithm also called the peeling algorithm uses a set of templates denoted by A_1, A_2, A_3, A_4, B_1 B_2, B_3 and B_4 shown in Fig.(3.4).

Matching or peeling is a thinning algorithm in which pixels are peeled (removed) from line edges until a one pixel wide line remains [30].

Pixels are removed by comparing each 1 valued pixel and its neighbors in the image with a set of templates.

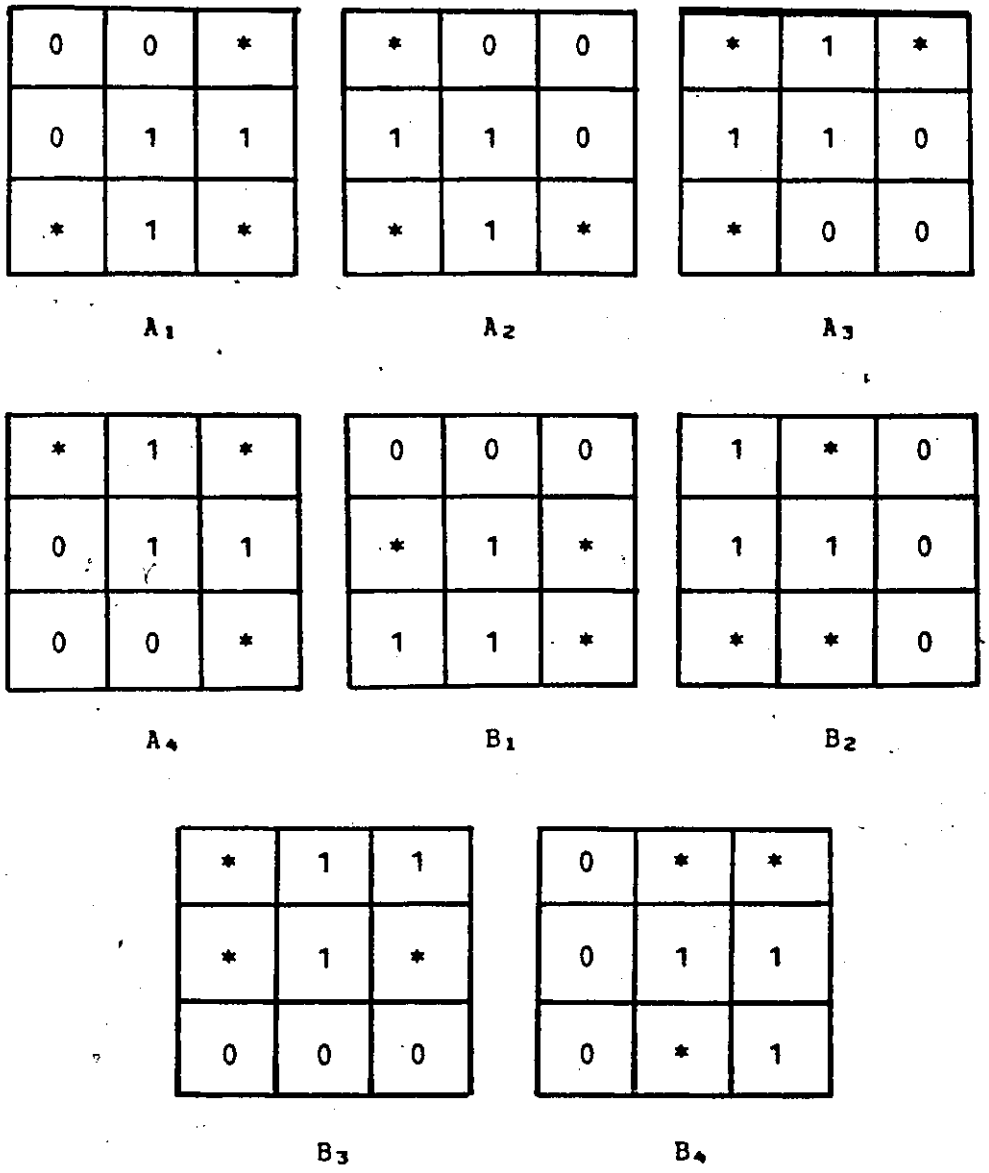


Figure 3.4: " Set of templates "; the value 1 represents the object value; the value 0 represents the background value and * represents don't care conditions (i.e can be either 1 or 0).

In the templates , zero must match 0 valued pixels ones must match 1 valued pixels and asterisks can match either 1 or 0 valued pixels in the image.

To begin for example template A_1 is compared with all 1 valued pixels and their neighbors in the image.

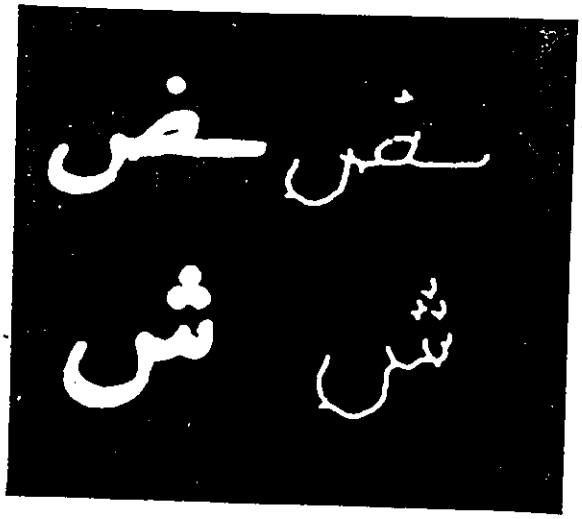
If the match is obtained , the corresponding central pixel of the image is deleted (changed to zero valued pixel).

After processing with template A_1 The process is repeated with template B_1, then with A_2 , B_2, A_3, B_3, A_4 and B_4 in that order forming a complete cycle. When no pixels are removed during the processing of a complete cycle , the procedure ends.

the filename of the program : temp1.for

3.2.3 RESULTS

Using the matching algorithm [30-31] in which the eight templates A_1,A_2,A_3,A_4,B_1,B_2,B_3,B_4 and only one image are used in the processing, we obtain a skeleton but not in the middle and with branches in many parts of the character . Results of using the above algorithm on Arabic characters are shown in Fig.(3.5).



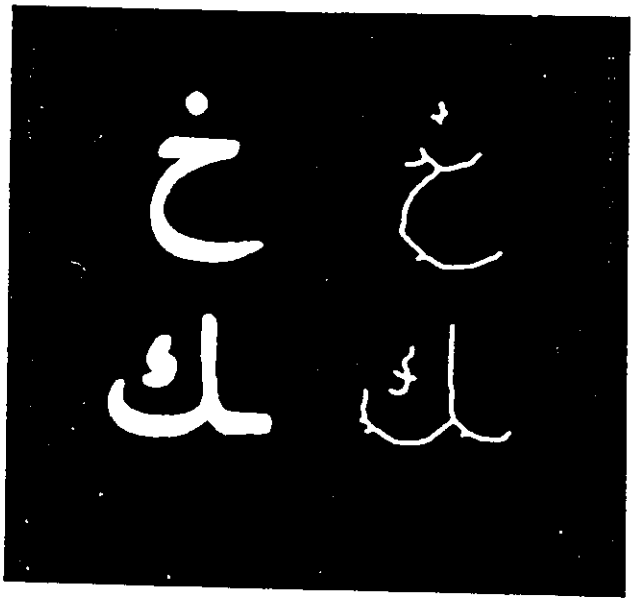


Figure 3.5: Skeletons obtained using the matching algorithm [30]

3.3 ALGORITHM #2 PARALLEL THINNING ALGORITHM [35].

3.3.1 INTRODUCTION

Usually different researchers have described various formal tests to prevent excessive erosion and to detect edge-points ,end-points and break-points.

3.3.2 NAGGACHE'S ALGORITHM

This algorithm is based on the above criteria .

For the algorithm we define the following .

1. $B(P_1)$: the number of nonzero neighbors of P_1

$$B(P_1) = P_2 + P_3 + P_4 + P_5 + P_6 + P_7 + P_8 + P_9$$

2. An edge-point as a dark point with at least one white 4-neighbors.
3. An end-point as a dark point with at the utmost one dark 8-neighbors.
4. The crossing number as

$$X(P_1) = \sum_{i=2}^{5} b_i$$

where

$b_i = 1$ if (point $P(2i-2)$ is white) and (either $P(2i-1)$ or $P(2i)$ is dark).

$b_i = 0$ otherwise

In any given pass , the input pattern is scanned row-wise from left to right and from top to bottom. A dark

point P_1 is flagged if all of the following 6 tests H1-H6 return the value true.

H1. $P_2 + P_4 + P_6 + P_8 \leq 3$ (P_1 is an edge-point).

H2. $B(P_1) \geq 2$ (P_1 is not an end-point)

H3. $N(P_1) \geq 1$ $N(P_1)$ is the number of unflagged at "tip of a thin line" or in approximatively circular subsets from being iteratively deleted.

H4. $X(P_1) = 1$ P_1 is not a break-point

H5. Either P_4 is unflagged

or $X_4(P_1) = 1$ $X_4(P_1)$ is the crossing number of P_1 if we temporarily assume that P_4 is white. This test prevents excessive erosion.

H6. Either P_6 is unflagged

or $X_6(P_1) = 1$ This test prevents excessive erosion as it is similar to H5.

At the end of the pass all the flagged points are deleted.

3.3.3 EXPLANATION OF THE ALGORITHM

The six operations (H1,...,H6) given above can be explained by the following [44].

To flag the point P, we must show that P is not an end-point , nor a break-point , and nor would its deletion cause excessive erosion. We show that below such a point P needs to compare the neighborhood of P with the four windows shown in Fig.(3.6).

If the neighborhood of P matches any one of the four windows, then the point P is not flagged. It should be noted that the points shown as x's and y's in the windows are " don't care " points (i.e their whiteness or darkness is immaterial). We now examine the four windows one by one , to justify why these are the windows to conduct the break-point , end-point and excessive erosion tests on P.

If the neighborhood of P matches any of the windows (a), (b) or (c) of Fig.(3.6), then two situations may occur :

1- If all x's are white, then P is an end-point.

2- If at least one of the x's is a dark point, then P is a break point. Thus in either of these cases, P should not be flagged.

Now let us examine window (d) of Fig.(3.6). If at least one x and at least one y are dark, then P becomes a break-point and thus it should not be flagged. For further analysis, let us assume for the time being that all x's are white, then there are eight possible configurations as shown in Fig.(3.7). Configurations W_1, W_2, and W_3 make P an end-point, and configurations W_4 makes P a break-point. The absence of conditions H5 and H6 could cause excessive erosion in slanting strokes of width 2; e.g Fig.(3.3a) being reduced to Fig.(3.8b). The importance of using these conditions is shown in Fig.(3.3c). It was stated by Naccache that in configurations W_5, W_6 and W_7 the point P_1 should not be flagged (found by experiment). In

configurations W_7 and W_8, the point P is in fact noise. Since the patterns have already been smoothed before they are fed to our algorithm, such noise is always removed and configurations W_7 and W_8 shall never exist at the beginning of the skeletonization process. If configuration W_7 were to occur in an intermediate stage of skeletonization, then P would be a spur due to a short tail in the original pattern (e.g., as in chromosomes). Such a point may, however, retain some shape information of the pattern and it must not be deleted. Similarly, if configuration W_8 were to occur in an intermediate stage of skeletonization, then it would be a singleton (isolated point); its deletion would totally erase the last remaining segment of the pattern. Therefore, in all of the above configurations P must not be flagged.

By symmetry, we extend our argument to the case in window (d) when all the y's are white and x's take on varying values of whiteness or darkness. Thus for window (d) also, x's and y's become " don't care " points.

1	0	x
0	P	x
x	x	x

(a)

x	x	x
0	P	x
1	0	x

(b)

x	0	0
0	P	1
x	0	0

(c)

x	x	x
0	P	0
y	y	y

(d)

Figure 3.6: If the neighborhood of a dark point P matches any of the four windows above the point P is not flagged ; x's and y's are don't care conditions.

The eight possible configurations that could exist in the window (d) of Fig.(3.6) are shown in Fig.(3.7).

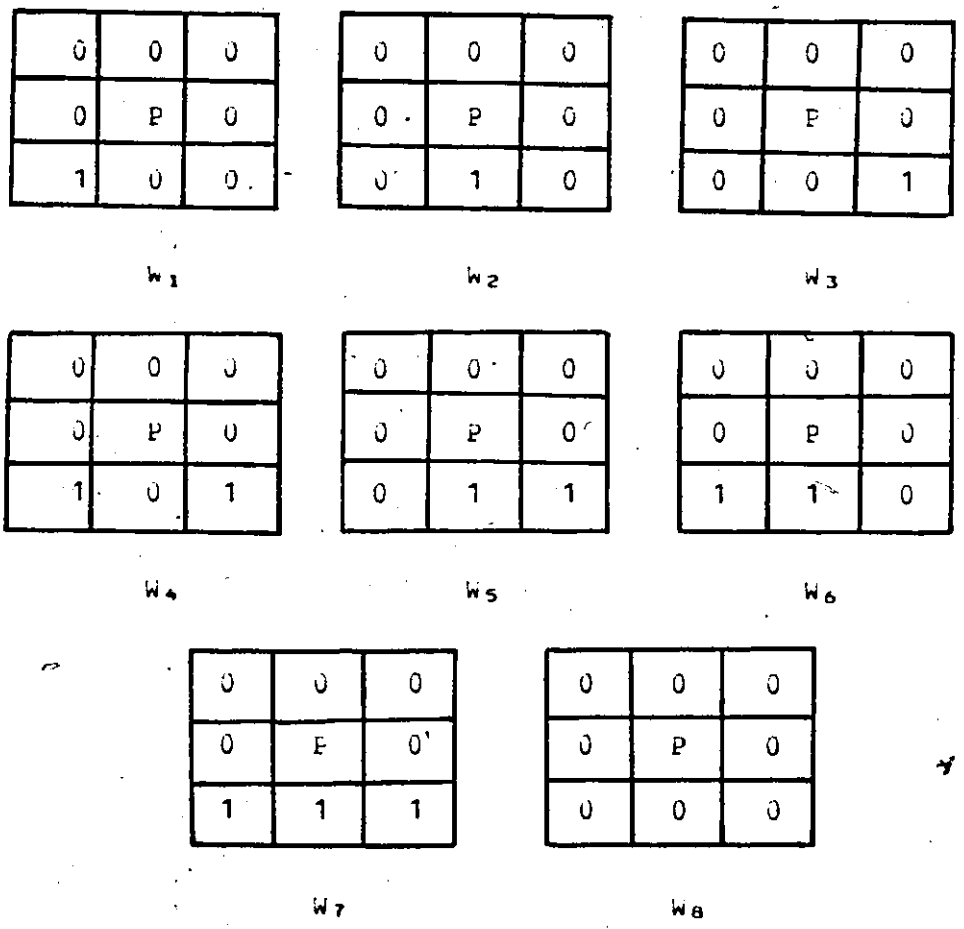


Figure 3.7: Eight windows derived from the window (d)

Fig.(3.8a) shows a slanting stroke of width 2. Fig.(3.8b) shows a skeleton obtained using the parallel thinning algorithm without conditions H5 and H6, and Fig.(3.8c) shows a skeleton obtained in the presence of conditions H5 and H6.

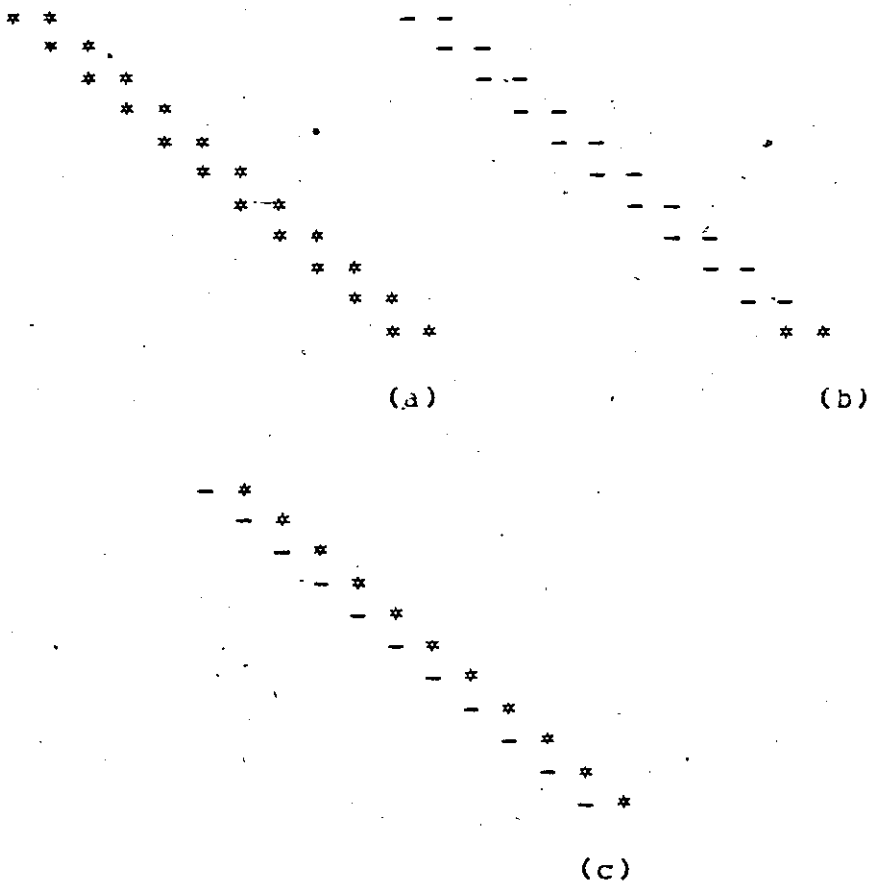


Figure 3.8: a- Slanting stroke of width 2 ; b- Skeleton obtained in the absence of conditions H5 and H6 ; c- Skeleton obtained in the presence of conditions H5 and H6.

3.3.4 RESULTS

The parallel thinning algorithm given by N.Naccache [35] in 1984 which conserves very well the shape of the original binary image has the following two problems.

1. It leaves extraneous pixels (branches) in the skeleton
2. The final skeleton is not connected

Results of using the above algorithm on Arabic characters are shown in Fig.(3.9).

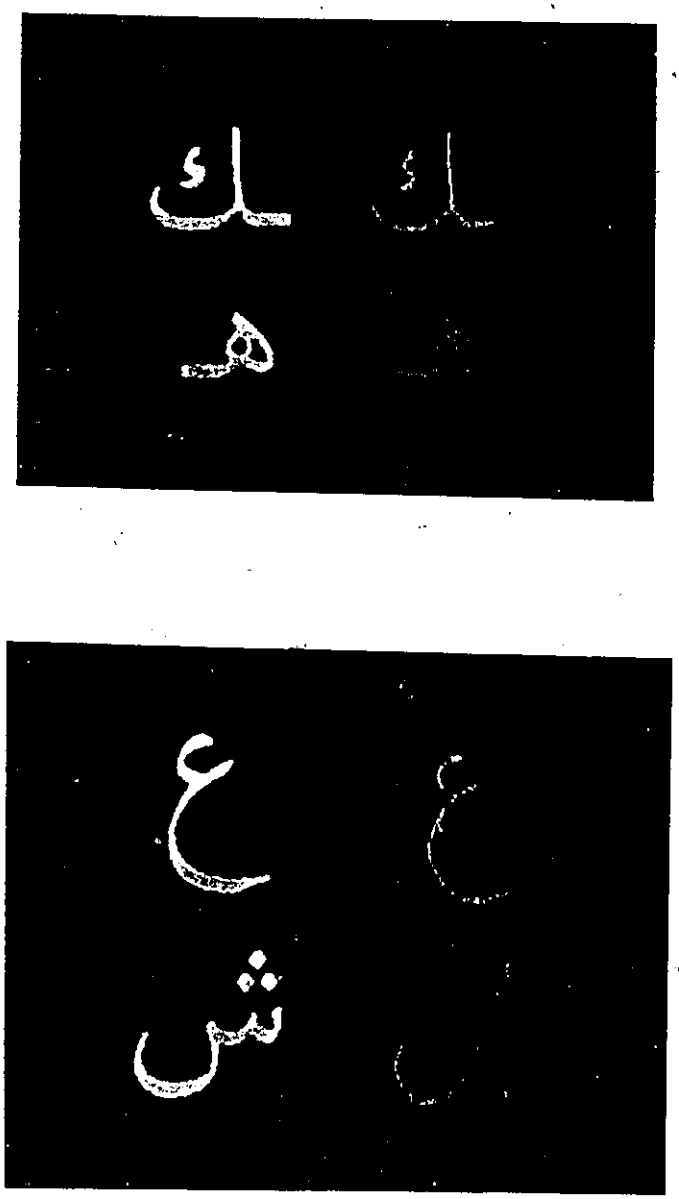


Figure 3.9: Skeletons obtained using Naccache's algorithm [35].

3.4 ALGORITHM #3 SIMPLIFIED VERSION OF HILDITCH [23]

3.4.1 DESCRIPTION :

A well known skeletonization approach is Hilditch's algorithm. Hilditch describes in detail the criteria that must be satisfied before a dark point of a pattern is deleted. However she didn't present her approach in a compact algorithmic form .

R.Stafanelli and A.Rosenfeld [23] presented a "simplied version " of Hilditch's apprcach as a formal algorithm.

A two-tone digitized picture is defined by a square matrix A ,where each element $a_{i,j}$ is either 1 or 0. It is supposed that the objects consist of those elements which have value 1.

Picture processing generally involves iterative transformations applied to the matrix A , where each transformed point depends on (i.e Boolean function of) a small set of neighboring points.

It is usually assumed that the neighbors of the point (i,j) are:

$(i,j+1),(i-1,j+1),(i-1,j),(i-1,j-1),(i,j-1)$

$(i+1,j-1),(i+1,j),(i+1,j+1)$ and the point

(i,j) itself as in Figure(3.10).

P_9 $(i-1,j-1)$	P_2 $(i-1,j)$	P_3 $(i-1,j+1)$
P_8 $(i,j-1)$	P_1 (i,j)	P_4 $(i,j+1)$
P_7 $(i+1,j-1)$	P_6 $(i+1,j)$	P_5 $(i+1,j+1)$

Figure 3.10: Designation of the nine pixels in a 3-by-3 window

Note that larger neighborhoods(5*5 and 4*7) have also sometimes been used [26-27] .The algorithm requires simple computations and makes use of a small (3*3) window.

3.4.2 ALGORITHM :

A first possible method for extracting the medial line (i.e the skeleton) of the figure consists of removing all the contour points of the figure except the ones that might belong to the line. The eight points P_2 , P_3 , P_4 , P_5 , P_6 , P_7 , P_8 and P_9 are known as the 8-neighbors of P_1 and the four points P_2 , P_4 , P_6 and P_8 are known as the 4-neighbors of P_1

In the algorithm, the contour point P_1 is deleted from the digital pattern if it satisfies the following tests.

1. $2 \leq B(P_1) \leq 6$

2. $A(P_1) = 1$

3. $P_2 * P_4 * P_6 = 0$

4. $P_2 * P_4 * P_8 = 0$

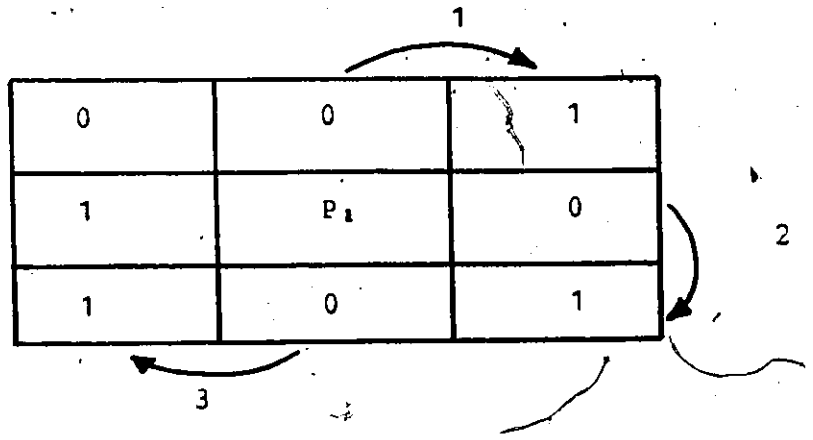
Where:

$A(P_1)$ is the number of 01 patterns in the ordered P_2, P_3, P_4, P_5, P_6, P_7, P_8, P_9 and P_2 that are the eight neighbors of P_1

$B(P_1)$ is the number of nonzero neighbors of P_1 that is:

$$B(P_1) = P_2 + P_3 + P_4 + P_5 + P_6 + P_7 + P_8 + P_9$$

If any condition is not satisfied then P_1 is not deleted from the picture. As an example of this is shown in Fig.(3.11) where $A(P_1) = 3$.



$A(P_1) = 3$

Figure 3.11: Counting the 01 pattern in the ordered set $P_2, P_3, P_4, P_5, P_6, P_7, P_8, P_9$ and P_2

3.4.3 RESULTS

The simplified version of Hilditch presented by Stefanelli and Rosenfeld [23] is described as follows :

a- A point P_1 is deleted from the pattern if :

a- $2 \leq B(P_1) \leq 6$

b- $A(P_1) = 1$

c- $P_2 * P_4 * P_6 = 0$

d- $P_2 * P_4 * P_8 = 0$

are satisfied. The algorithm may not retain the shape properties of the original pattern and sometimes the presence of extraneous pixels (branches) in the final skeleton . Results of using the above algorithm with procedure (a) on Arabic characters are shown in Fig.(3.12).

The filename of the program : ALGSUB10.for

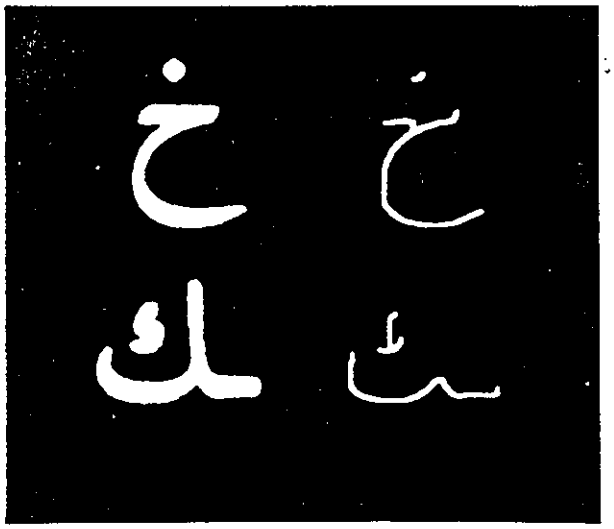
b- A point P_1 is flagged (labeled with another value rather than 0 or 1) in each pass and at the end of each pass all the flagged points are deleted.

The algorithm in this case yields the following results:

1. We obtained a skeleton but not in the middle.
2. The algorithm leaves a few extraneous pixels and sometimes the shape of the original image is not preserved.

Results of using the above algorithm with procedure (b) on Arabic characters are shown in Fig.(3.13).

The filename of the program is : ALGSUB1.for



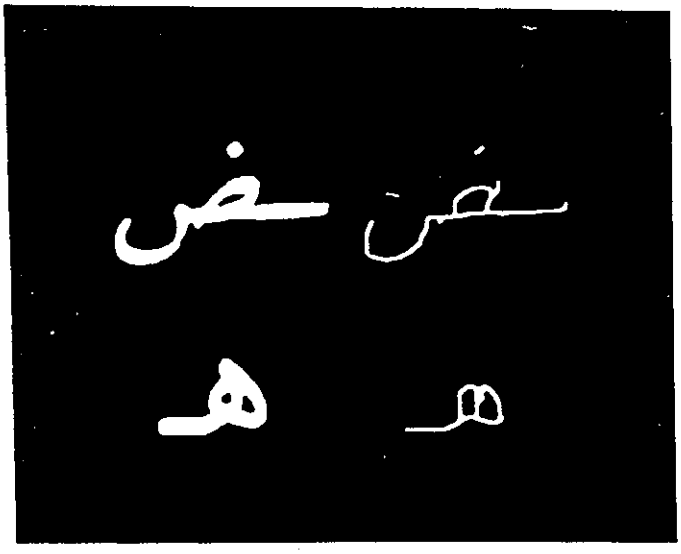


Figure_3.12: Skeletons obtained using the " Simplified version of Hilditch when the central point P1 is deleted.

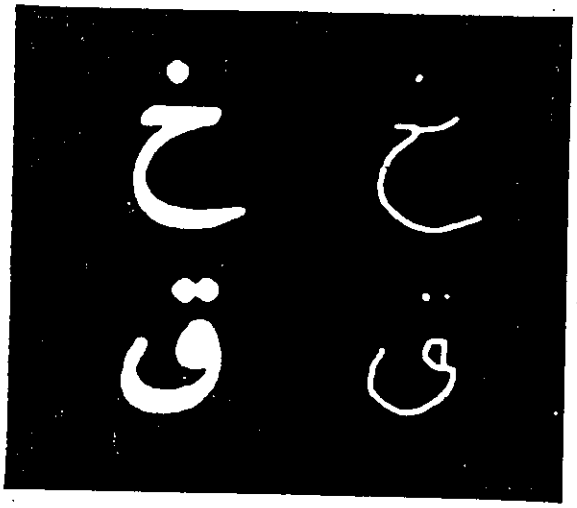


Figure 3.13: Skeletons obtained using the " Simplified version of Hilditch when the central point P_1 is flagged.

3.5 ALGORITHM #4 SEQUENTIAL THINNING ALGORITHM [26-27]

3.5.1 DESCRIPTION

The sequential thinning algorithm presented in [27] consists of three steps :

1. Sequential application of a modified distance transform ;
2. linking algorithm ; and
3. thinning algorithm.

The block diagram shown in Figure(3.14) indicates the image analysis process for extraction of shape and position information from the binary image. Given a thresholded binary image (see APPENDIX - A -), followed by the distance transform , an initial skeleton is obtained by applying the first or the second equation of the new selection rules, a linking algorithm is applied immediately after the new selection rules to achieve a connected skeleton, and because the resultant skeleton is not one pixel thin everywhere, a thinning procedure is needed to obtain the final skeleton.

In the following sections each block is explained in detail.

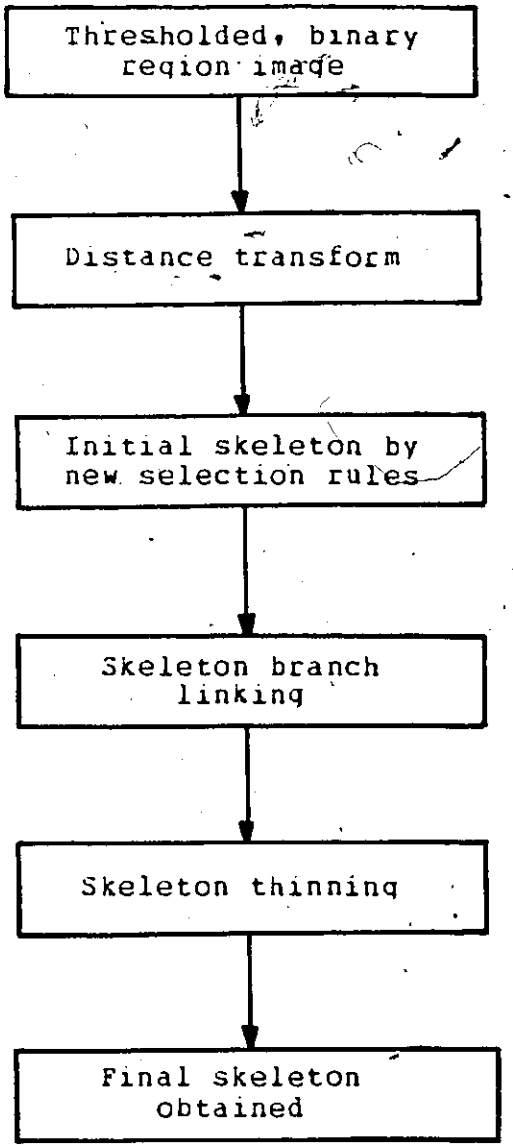


Figure 3.14: The block diagram

3.5.2 ALGORITHM

3.5.2.1 MODIFIED DISTANCE TRANSFORM :

The modified distance transform includes the distance transform and the new set of selection rules for picking up skeleton elements.

DISTANCE TRANSFORM (DT) [40] :

Distance transform is used by many researchers in the determination of the skeletons because the processing time is very fast.

Given a digitized picture whose elements have only the values 0 or 1, it is desirable to construct a distance transform of the picture in which each element has an integer value equal to its distance from the set of 0's , (it is assumed that the set of o's is nonempty). Thus in particular , the 0's remain unchanged since they are at zero distance from themselves, the 1's which are horizontal or vertical neighbors of 0's also remain unchanged , the 1's which are horizontal or vertical neighbors of such 1's become 2's and so on.

The distance transform utilized in [26] is substituted by the distance transform explained in [40]. It can be performed using the following method :

1- On the first pass the image is scanned row-wise from the upper left hand corner to the lower right hand corner . Each 1 valued pixel in the image is assigned a new valued according to :

$$D(row,column) = \min(D(row-1,column),$$
$$D(row,column-1))+1.$$

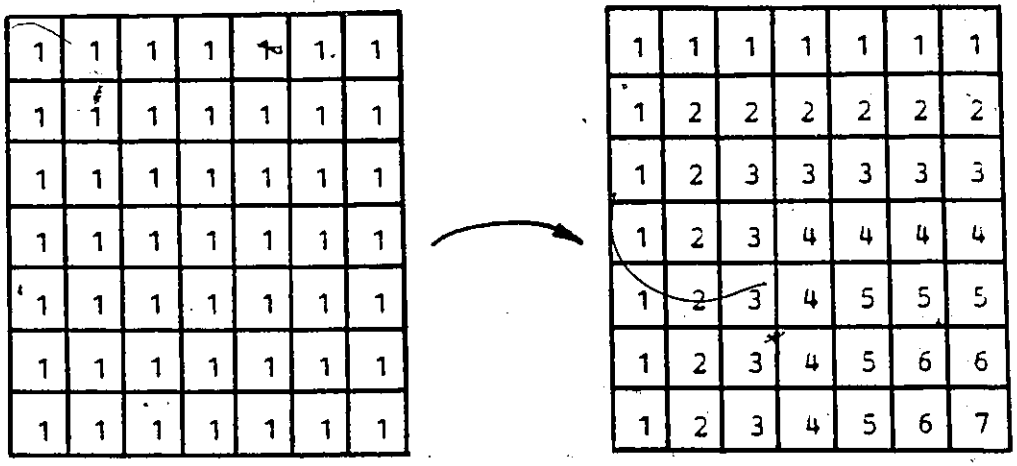
This calculates each pixel's distance from the left to top side of a line.

2- On the second pass, the image is scanned row-wise from the lower right hand corner to the upper left hand corner. Each nonzero valued pixel is assigned a new valued pixel according to :

$$D(row,column) = \min(D(row+1,column)+1,$$
$$D(row,column+1)+1,$$
$$D(row,column)).$$

As an example of this is shown in Figure(3.15)

1- The first pass



2- The second pass

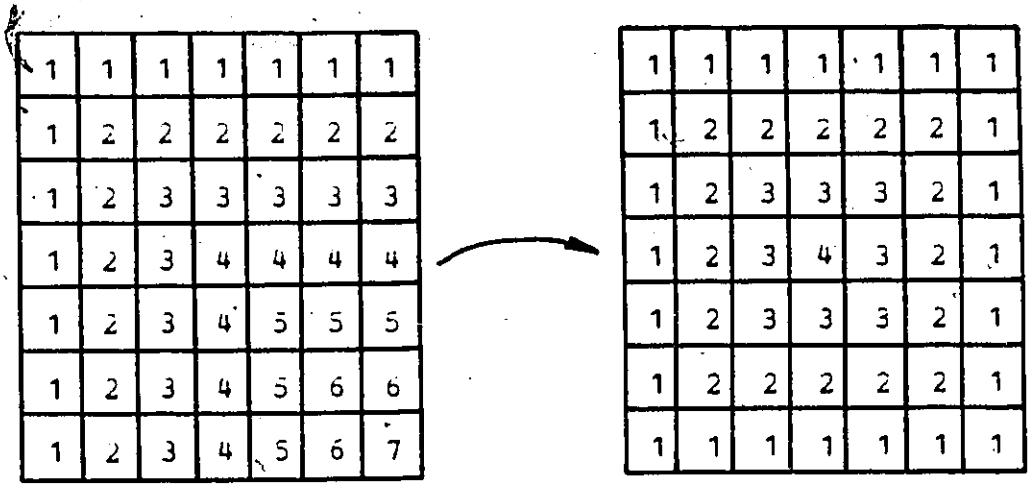


Figure_3.15: Example of using the distance transform [40] given above

A simpler distance transform than the ones explained in [25] and [40] is given as follows :

Let S be a subset of a digital image , we sometimes refer to the point of S as 1's and to the point not in S as 0's.

For any point P of S , let d(p) denote the distance from the point P to the nearest point not in S.

The distance between two points (x,y) and (u,v) can be obtained by taking the minimum distance between the points (x,y) and (u,v) in the four directions (north , south , east and west).

Note that (x,y) is a point in S (it means 1 valued pixel) and (u,v) a point not in S (it means 0 valued pixel).

Using this method we obtained the same result as before (see Fig.(3.15).

NEW SELECTION RULES:

The goal of the new selection rules is to generate an initial skeleton , also to select more pixels as skeleton elements to achieve a more connected skeleton at the outset of the skeletonization.

There are two ways to perform the new selection rules.

The first one selects the skeleton using the cross neighborhood of length one of each pixel according to equation (a) :

Equation (a) can be written as :

if (D(i,j)+1 > D(i,j-1) , D(i,j)+1 > D(i-1,j) ,

D(i,j)+1 > D(i,j+1) and D(i,j)+1 > D(i+1,j)

then D2(i,j) = D(i,j)

otherwise D2(i,j) = 0.

The second one selects the skeleton using an extended cross neighborhood of length two according to equation (b) as shown in Fig.(3.16).

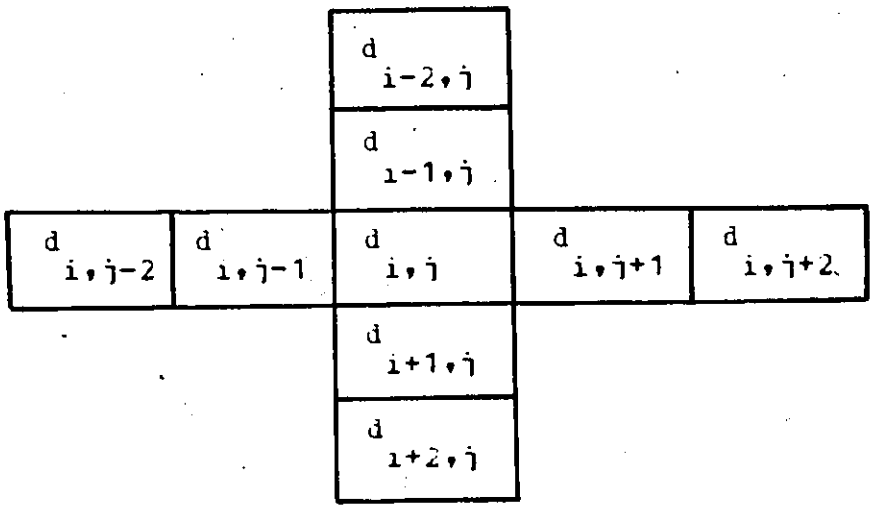


Figure 3.16: Cross neighborhood for MDT selection rules

Equation (b) can be written as :

if (D(i,j)-D(i,j-2) + D(i,j)-D(i-2,j) +

D(i,j)-D(i+2,j)+ D(i,j)-D(i,j+2) > 2)

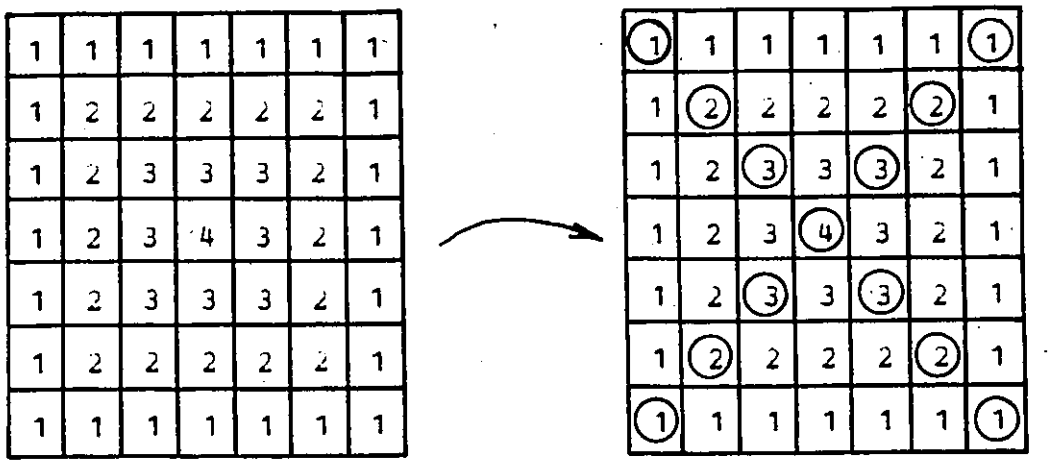
then D2(i,j) = D(i,j)

otherwise D2(i,j) = 0

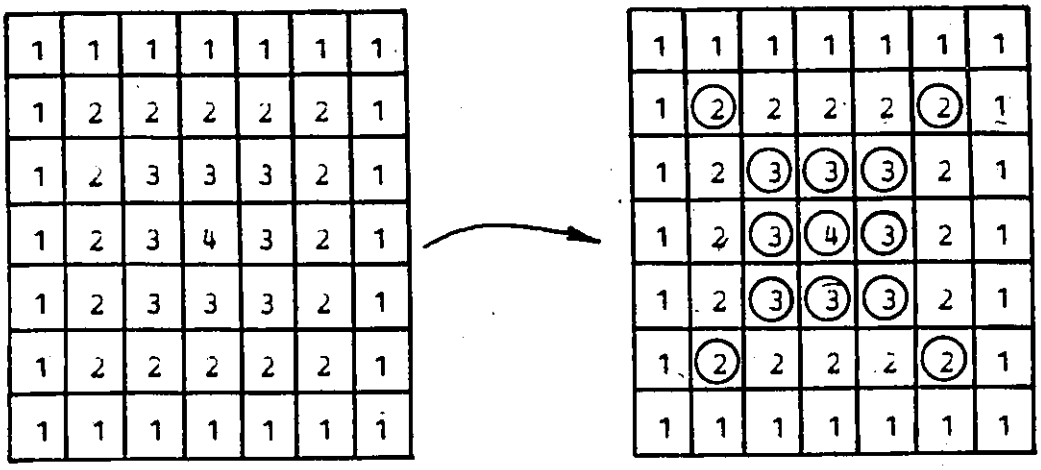
Note that by using the second equation we produce a skeleton which is more connected than the skeleton obtained by using the first equation.

D(i,j) represents the distance transform and D2(i,j) represents the initial skeleton obtained using the new selection rules.

An example of using equations (a) and (b) is shown in Fig.(3.17).



(a)



(b)

Figure 3.17: a- Skeleton obtained using equation (a) ; b- Skeleton obtained using equation (b)

3.5.2.2 LINKING ALGORITHM:

A linking algorithm is applied immediately after the new selection rules to achieve a connected skeleton.

The linking algorithm links the gaps produced by the new selection rules so that a connected skeleton can be obtained.

The algorithm can be divided into two steps :

1. Linearity test
2. Linking process

For the purpose of minimizing the processing time, the linearity test is introduced to detect a possible linearly connected skeleton segment within the 3 by 3 window centered at the current pixel . If the linearly connected skeleton segment can be detected in any horizontal , vertical, or diagonal direction within the 3-by-3 window , the linking process is bypassed, and time otherwise required to accomplish the linking process is saved. If the linearity test fails to detect any linearly connected segment within the 3-by-3 window, pixels are tested extending over a 7-by-4 window for the linking process.

1. Linearity test :

The linearity test is operated over the 3-by-3 window centered at the current pixel $P_{i,j}$. It is only performed when the DT value of any image point is greater than zero , i.e when an image point belongs to a skeleton. A linearly connected skeleton segment is defined as :

(i) Having three elements, in the same row or column,

or any of two diagonal axes of the window.

(ii) All the DT values of the three elements must be greater than zero.

Without the linearily test, the linking process described below would itself still do the job.

2- Linking process :

The linking process is operated over a 7-by-4 window as shown in Fig.(3.18). The window is moved in a similar fashion as the 3-by-3 window. The current pixel, Pi,j is located at the middle element of the leftmost column of the 7-by-4 window.

The linking area is subdivided into five regions (see Fig.(3.18)): North (N) ; Northeast (NE) ; East (E) ; Southeast (SE) ; and South (S) . The linking operations are sequentially executed in clockwise directions from the North region to the South region. The NE and the SE regions are 3-by-3 windows; the N and S are 3-by-1 windows; and the E region is 1-by-3 window. In each region , if the element closest to the current pixel (or one of its 8-neighbors) is the skeleton element, the linking operations are bypassed and proceed to the next region in sequence. In the N, NE, ane E regions, the linking operations try to link two connected components into one connected component ; but in the SE and S regions, they try to reduce the gap between two connected components from two pixels to a single pixel

separation. Once a connected component is obtained or a gap has been reduced in a region, the linking process is terminated for that particular pixel, and the linking window is moved to the next pixel.

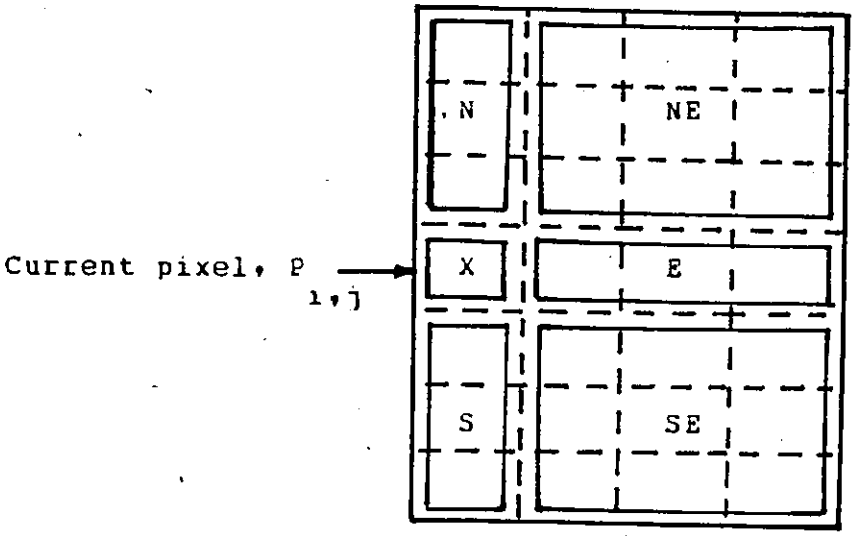


Figure 3.18: 7-by-4 neighborhood window for skeleton linking

The resultant skeleton is connected , but not pixel thin everywhere. Therefore a thinning procedure is needed to obtain the skeleton (i.e one pixel thick)

3.5.2.3 THINNING ALGORITHM:

All selected skeleton elements are actually in distance transformed values. These elements are converted into 1's, where non-skeleton elements, even those with distance transformed values greater than 0 are converted into 0's.

The proposed algorithm is to test all the combinations to determine whether the current pixel P_o can be removed or not. The window is moved from the upper left corner to the lower right corner of the image. If the connectivity of all neighboring skeleton pixels in the window is preserved after removing the center pixel from the window, the center pixel is permanently removed (converted to zero). However, if the window contains only one neighbor pixel, the center pixel is not removed. If it contains no neighboring pixel, the center pixel (an isolated pixel) is removed.

The thinning algorithm proposed according to the 8-connected definition, consists of the following six operations.

1. STORING:

The presence or absence of any neighbors Pi for $1 \leq i \leq 8$ is represented by 1 or 0 respectively , for $1 \leq i \leq 8$ in bite T .(T_8 is the most significant bit in the left most position), see Figure(3.19).

2. MASKING:

Bits T_2 , T_4 , T_6 and T_8 of byte T corresponding to the four corners of the window are masked out (turned to zero) and the result is stored in byte TM.

3. SHIFTING:

The contents of TM are shifted left by one bit (with end-around carry) and the result is represented by Q.

4. OR.ing

The contents of T and Q are OR'ed together and stored into TR.

5. COUNTING:

The bits in every pair of consecutive positions (including end-around carry) in TR are compared. For every change from 1 to 0 in the pair , a change counter $N_{1\rightarrow 0}(TR)$ is incremented. The total number of 1's in TR ($N_1(TR)$) is also computed.

6. DECISION MAKING:

If the change counter from step 5 , $N_{1\rightarrow 0}(TR)$ is greater than 1, the current pixel P_0 is not removed. If $N_{1\rightarrow 0}$ TR) contains 1, and the total number of 1's in TR ($N_1(TR)$) is also 1 the current pixel is not removed ; otherwise the current pixel is removed.

Figure(3.19) presents a 3-by-3 neighborhood with pixel definitions

P_8	P_1	P_2
P_7	P_0	P_3
P_6	P_5	P_4

Figure 3.19: P_0 : current pixel ; $P_1, P_2, \ldots, P_9$: eight-neighbors pixel ; P_1, P_3, P_5, P_7 : four-neighbors pixel

An example of the thinning algorithm is presented in the following.

1	0	1
0	1	0
1	0	1

(A)

0	1	1
0	1	1
0	1	0

(B)

	T_8	T_7	T_6	T_5	T_4	T_3	T_2	T_1	T_8	T_7	T_6	T_5	T_4	T_3	T_2	T_1
1. T :	1	0	1	0	1	0	1	0	0	0	0	1	0	1	1	1
2. TM:	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1
3. Q :	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0
4. TR:	1	0	1	0	1	0	1	0	0	0	1	1	1	1	1	1

5. $N_{1\rightarrow 0}(TR) = 4$; $N_{1}(TR) = 4$ | $N_{1\rightarrow 0}(TR) = 1$; $N_{1}(TR) = 6$

6. P_o is not removed | P_o is removed

3.5.3 RESULTS

As a result the algorithm (the explanation given above) preserves very well the shape of the original image and its processing time is very short (at the utmost 30 seconds) . The disadvantage is the appearance of the disconnectivity where the linking algorithm fails to do it in characters as shown in Figs.(3.20) and (3.21).

1. Using the first equation of the new selection rules the filename of the program is : D.for
2. Using the second equation of the new selection rules the filename of the program is : D1.for

Results of using the above algorithm on Arabic characters either by using the first or the second equation of the new selection rules are shown in Fig.(3.20) and (3.21).

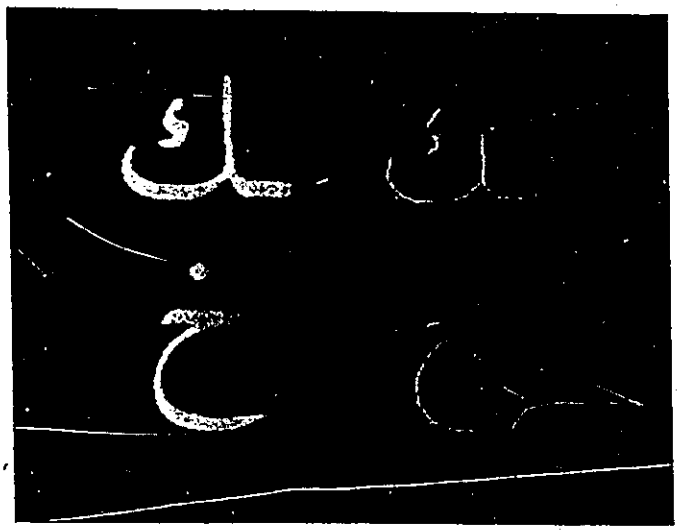


Figure_3.20: Skeletons obtained using the sequential thinning algorithm by using the first equation of the new selection rules.

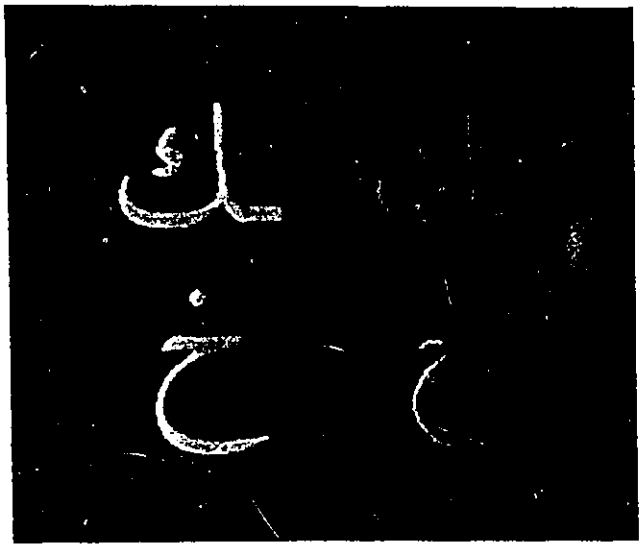


Figure_3.21: Skeletons obtained using the sequential thinning algorithm by using the second equation of the new selection rules.

3.6 CONCLUSION

Four parallel thinning algorithms are presented, implemented and applied to Arabic fonts namely the Matching algorithm, the " Simplified version of Hilditch ", Naccache's algorithm and the sequential thinning algorithm. We have shown the problems of each algorithm when applied to Arabic fonts. These problems lead us to develop new thinning algorithms as it will be explained in the next section.

Chapter IV

EFFECTIVE THINNING AND THICKENING ALGORITHMS WITH APPLICATION TO ARABIC FONTS

4.1 INTRODUCTION :

In picture analysis it is often convenient to deal with a stick line version (skeleton) of binary images. A thinning algorithm is considered useful for a particular application if it maintains the connectivity, conserves the shape of the original image and does not leave extraneous pixels (branches) in the final skeleton . Two parallel thinning algorithms are developed. One is based on local operations to detect edge-points, end-points and break points, and it consists of four sub-iterations in which each one is divided into four conditions. The other is a modified approach of a matching algorithm [30] in which a set of eight templates and two images (the current image and the working image) are used in the processing. Both algorithms maintain the connectivity, conserve the shape of the original image and do not leave extraneous pixels within a 3-by-3 window to produce a pixel thin skeleton from a connected thick image. Also a thickening algorithm is presented.

Experimental results show that these methods are very effective for thinning and thickening of Arabic characters.

4.2 NEW PARALLLEL THINNING ALGORITHM :

4.2.1 INTRODUCTION

The parallel thinning algorithm is developed to investigate the problems discussed in Chapter III. It is based on local operations to detect edge-points, end-points and break-points (i.e that the point P_1 is a condidate for deletion if it is an edge-point, not an end-point and not a break-point). Also the algorithm consists of four sub-iterations

4.2.2 ALGORITHM

In each pass the input pattern is scanned row-wise from the upper left hand corner to the lower right hand corner.

In each pass a dark point P_1 is flagged if at least one of the following four sub-iterations is satisfied .

Note that each sub-iteration is divided into four conditions.

The first sub-iteration

1. a- $2 \leq B(P_1) \leq 6$

   b- $A(P_1) = 1$

   c- $P_2 * P_4 * P_6 = 0$

   d- $P_4 * P_6 * P_8 = 0$

The second sub-iteration

2. a- $2 \leq B(P_1) \leq 6$

   b- $A(P_1) = 1$

   c- $P_2 * P_6 * P_8 = 0$

   d- $P_4 * P_6 * P_8 = 0$

The third sub-iteration

3. a- $2 \leq B(P_1) \leq 6$

b- $A(P_1) = 1$

c- $P_2 * P_4 * P_8 = 0$

d- $P_2 * P_6 * P_8 = 0$

The fourth sub-iteration

4. a- $2 \leq B(P_1) \leq 6$

b- $A(P_1) = 1$

c- $P_2 * P_4 * P_6 = 0$

d- $P_2 * P_4 * P_8 = 0$

At the end of each pass all the flagged pixels are deleted.

4.2.3 EXPLANATION OF THE FOUR SUB-ITERATIONS :

We begin by explaining the first and the second conditions because they are the same for all the sub-iterations , then conditions (c) and (d) for each sub-iteration are explained.

1- In condition (a) the end-points of the skeleton line are preserved (this means the condition verifies that P_1 is an edge-point and not an end-point).

Note that an edge-point is a dark point that has at least one white four-neighbors and an end-point is a dark point that has at most one dark eight-neighbors.

2- Condition (b) verifies that the point P_1 to be deleted is not a break-point (i.e its deletion must not break the connectedness of the original pattern).

3. Conditions (c) and (d) for the first sub-iteration :

$$P_2 * P_4 * P_6 = 0$$

$$P_4 * P_6 * P_8 = 0$$

In condition (3) the point P_1 which has been removed might be an east border point or south border point or north- west corner point

The solutions to the set of equations :

$$P_4 = 0$$

or

$$P_6 = 0$$

or

$$P_2 = 0 \text{ and } P_8 = 0$$

4. Conditions (c) and (d) for the second sub-iteration

$$P_2 * P_6 * P_8 = 0$$

$$P_4 * P_6 * P_8 = 0$$

In condition (4) the point P_1 which has been removed might be a south border point or a west border point or a north-east corner point.

The solutions to the set of equations are :

$$P_6 = 0$$

or

$$P_8 = 0$$

or

$$P_2 = 0 \text{ and } P_4 = 0$$

5. Conditions (c) and (d) for the third sub-iteration

$$P_2 * P_4 * P_8 = 0$$

$$P_2 * P_6 * P_8 = 0$$

In conditions (5) the point P_1 which has been removed might be a north border point or a west border point or a south-east corner point.

The solutions to the set of equations are :

$P_2 = 0$

or

$P_8 = 0$

or

$P_4 = 0$ and $P_6 = 0$

6. Conditions (c) and (d) for the fourth sub-iteration

$$P_2 * P_4 * P_6 = 0$$

$$P_2 * P_4 * P_8 = 0$$

In condition (6) the point P_1 which has been removed might be a north border point or an east border point or a south-west corner point.

The solutions to the set of equations are :

$P_2 = 0$

or

$P_4 = 0$

or

$P_6 = 0$ and $P_8 = 0$

North

	P_2	
P_8	P_1	P_4
	P_6	

West East

South

Figure 4.1: Points under consideration and their locations

The above algorithm can also be implemented using an alternative method as follows.

In conditions (c) and (d) for each sub-iteration the following eight templates can be created as shown in Fig.(4.2).

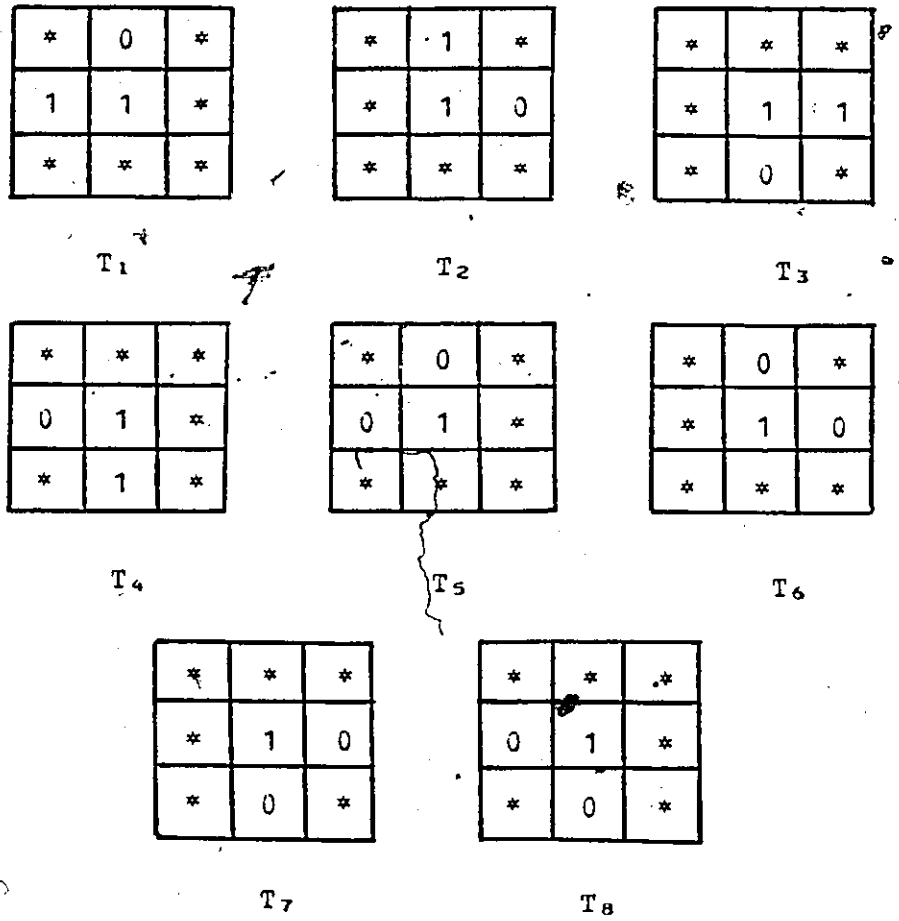


Figure 4.2: Set of templates used in the processing: The value 1 corresponds to the object, the value 0 corresponds to the background and * can be either 1 or 0.

If the two conditions

$$1.\ 2 \leq B(P_1) \leq 6$$

$$2.\ A(P_1) = 1$$

are satisfied and at least one of the eight templates T_1, T_2, T_3, T_4, T_5, T_6, T_7 and T_8 given above is matched with the pixels in the given image then the central pixel P_1 is flagged .

In each pass the point P_1 is flagged if at least one of the four sub-iterations is satisfied (first procedure) or the match is obtained (second procedure). At the end of each pass all flagged points are deleted.

The processus is repeated until no more pixels are deleted from the pattern, thus the final skeleton is obtained.

the filename of the program : alqsub4.for

4.2.4 RESULTS

The above algorithm either by using the first procedure or the second one yields excellent results with respect to the shape of the original image , connectivity and contour noise immunity.

Note that this algorithm is applied to Arabic characters and the results obtained are excellent as shown in Figs.(4.4) and (4.5).

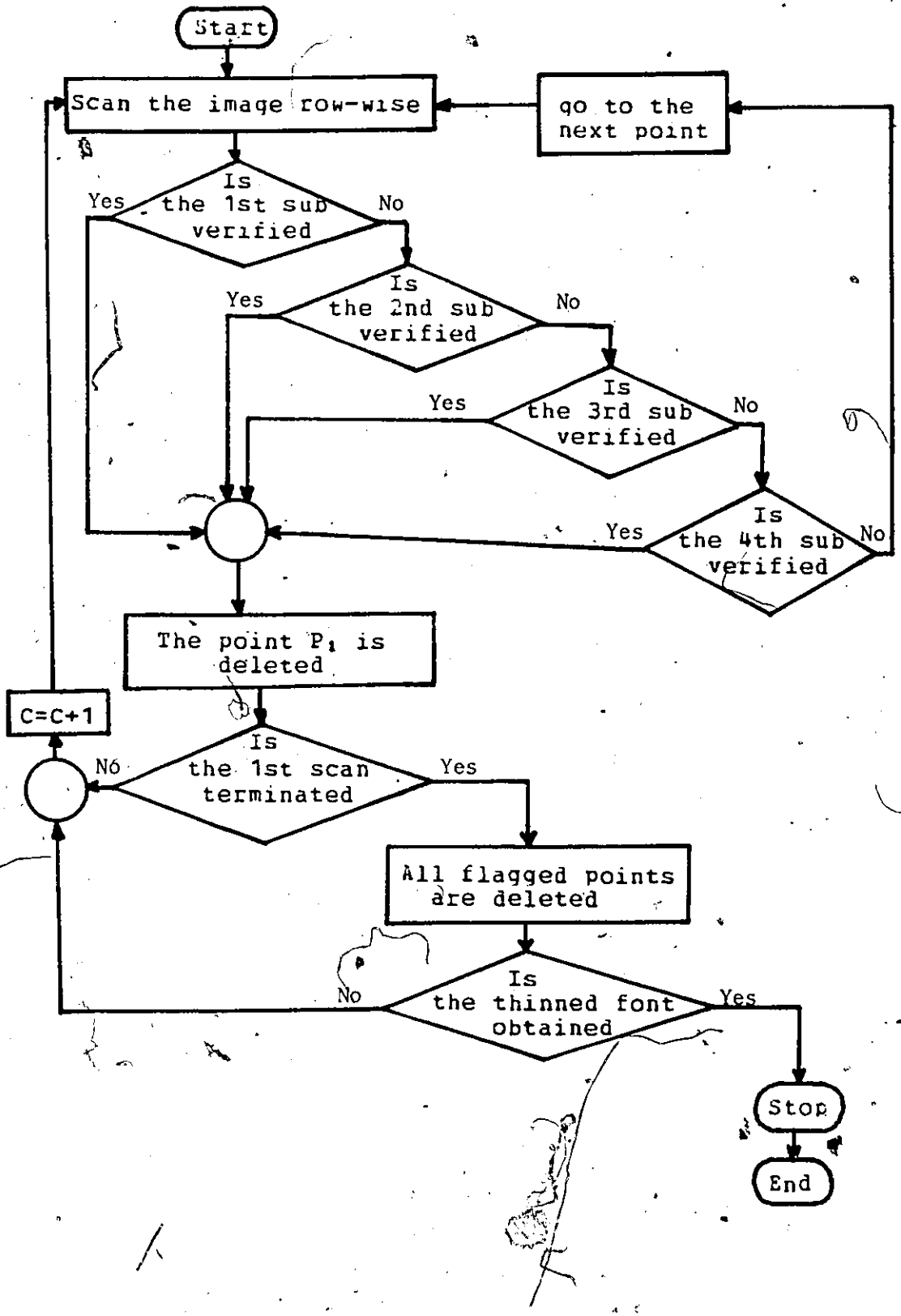
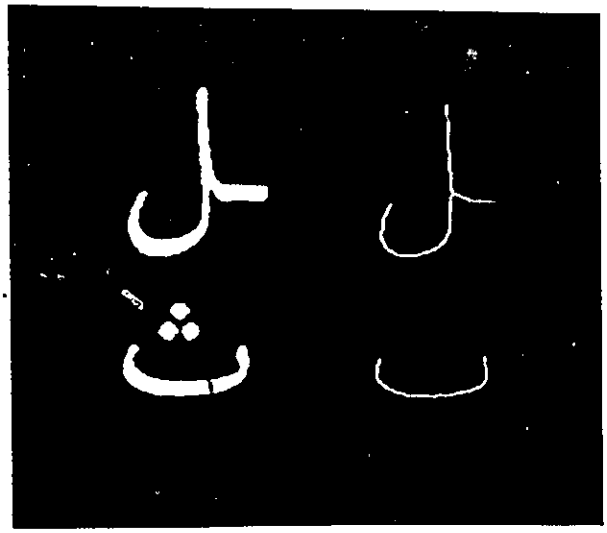


Figure 4.3: Flowchart of the new parallel thinning algorithm



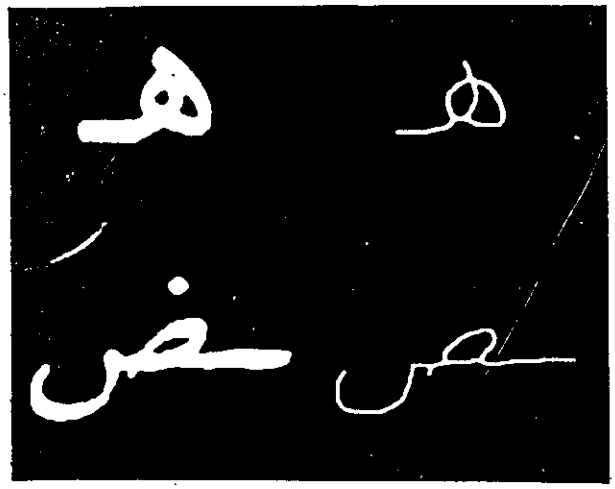


Figure_4.4: Skeletons obtained using the new parallel thinning algorithm

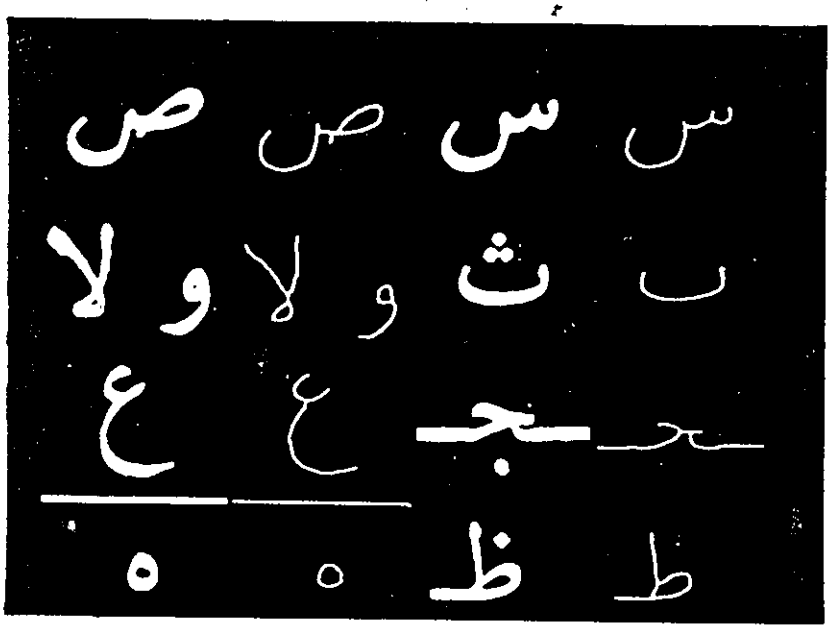


Figure 4.5: Skeletons obtained using the new parallel thinning algorithm

4.3 THE MODIFIED APPROACH OF MATCHING ALGORITHM :

4.3.1 INTRODUCTION:

In image processing, not only the development of new techniques but also the improvement of existing techniques are important. Development of both a parallel algorithm and a sequential algorithm for each technique is desirable, because image processing might be performed by a general purpose sequential computer or parallel processing dedicated hardware.

4.3.2 ALGORITHM :

The modified approach of a matching algorithm [30] is another parallel thinning algorithm in which a set of eight templates [30-31] denoted by A1, A2, A3, A4, B1, B2, B3, B4 (see Fig.(3.4)) and two images (a current image for which templates are compared and a working image of the same size which is updated when templates are matched) are used in the processing.

Pixels are removed by comparing each 1 valued pixel and its neighbors in the current image with a set of templates. In the templates zeros must match 0 valued pixels , ones must match 1 valued pixels and asterisks can match either 1 or 0 valued pixels in the current image.

Initially the current image and the working image are identical copies of the original input image.

To begin , template A_1 is compared with all 1 valued pixels and their neighbors in the current image . If the match is obtained, the corresponding central pixel of the working image is deleted (changed to a 0 valued pixel).

After processing with template A_1 , the current image is discarded and the working image becomes the new current image , and the new working image is obtained by copying the new current image .

The process is repeated with templates B_1, A_2, B_2, A_3, B_3, A_4 and B_4 in that order forming a complete cycle. When no pixels are removed during the processing of the complete cycle , the procedure ends.

As a result of this method , the shape of the original image is very well preserved and has good connectivity . The algorithm leaves very few extraneous pixels only on the dots of the characters as shown in Fig.(4.8).

the filename of the program : temp2.for

To avoid this problem (presence of extraneous pixels on the dots of the characters) we divided the character into two parts ; a character without dots and dots alone. Each part is processed separately.

To separate the dots from the character the following methods can be used :

1- Follow the border using the border following algorithm (see APPENDIX - B -).

2- The minimun border is the dot and the maximum border is the character without dots.

3- Find the location of the dots, which means find :

I_{min} , I_{max} , J_{min} and J_{max} as shown in Fig.(4.5).

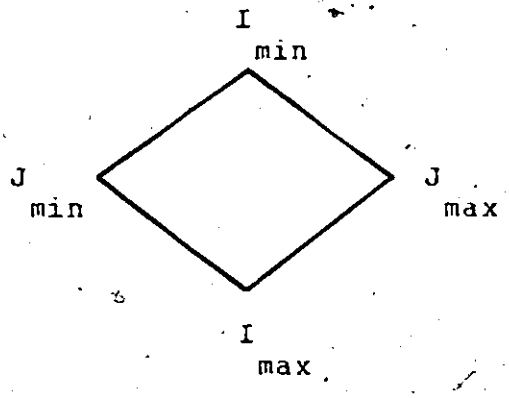


Figure 4.6: The location of the dot

4- Create a window depending of the size of each dot .

5- Store it in another array

6- Steps 3-5 are repeated for other dots.

An example of separating the dots from the characters is shown in Fig.(4.3).

After the separation the next step is to process each part separately

a- Character without dots : This process was done by the matching method using eight templates and two images (the working image and the current image).

b- The dots : For this case one of the following two methods can be used.

1. By the determination of the coordinates I_{min}, I_{max}, J_{min} and J_{max}, the dot can be replaced by the point (I,J) at the center which can be calculated as:

$$I = \frac{I_{min} + I_{max}}{2}$$

$$J = \frac{J_{min} + J_{max}}{2}$$

2. The second method consists of iteratively executing many passes over the pattern, where in each pass object points which lie on the boundary are changed to background points.

For the first pass we scan the image row-wise from the upper left corner to the bottom right corner and for every change from 0 to 1 (i.e from the background to the object pixel) or 1 to 0 (i.e from the object pixel to the background pixel), we label the object with value 2 which means that the value (2) will belong to the background after the second pass.

For the second pass we scan the image column-wise from the upper left corner to the bottom right corner and also for every change from 0 to 1 or 1 to 0 we label the object by the value 2 .

Once the second pass is terminated all the pixels with label (2) are then changed to the value 1 which is the value of the object.

The process is repeated with in the same manner as explained above until we reach the point.

Once the processing is terminated for the two parts the last step is to add them to one which is our final skeleton.

the filename of the program : tempwind.for

The flowchart of this method is shown in Fig.(4.7).

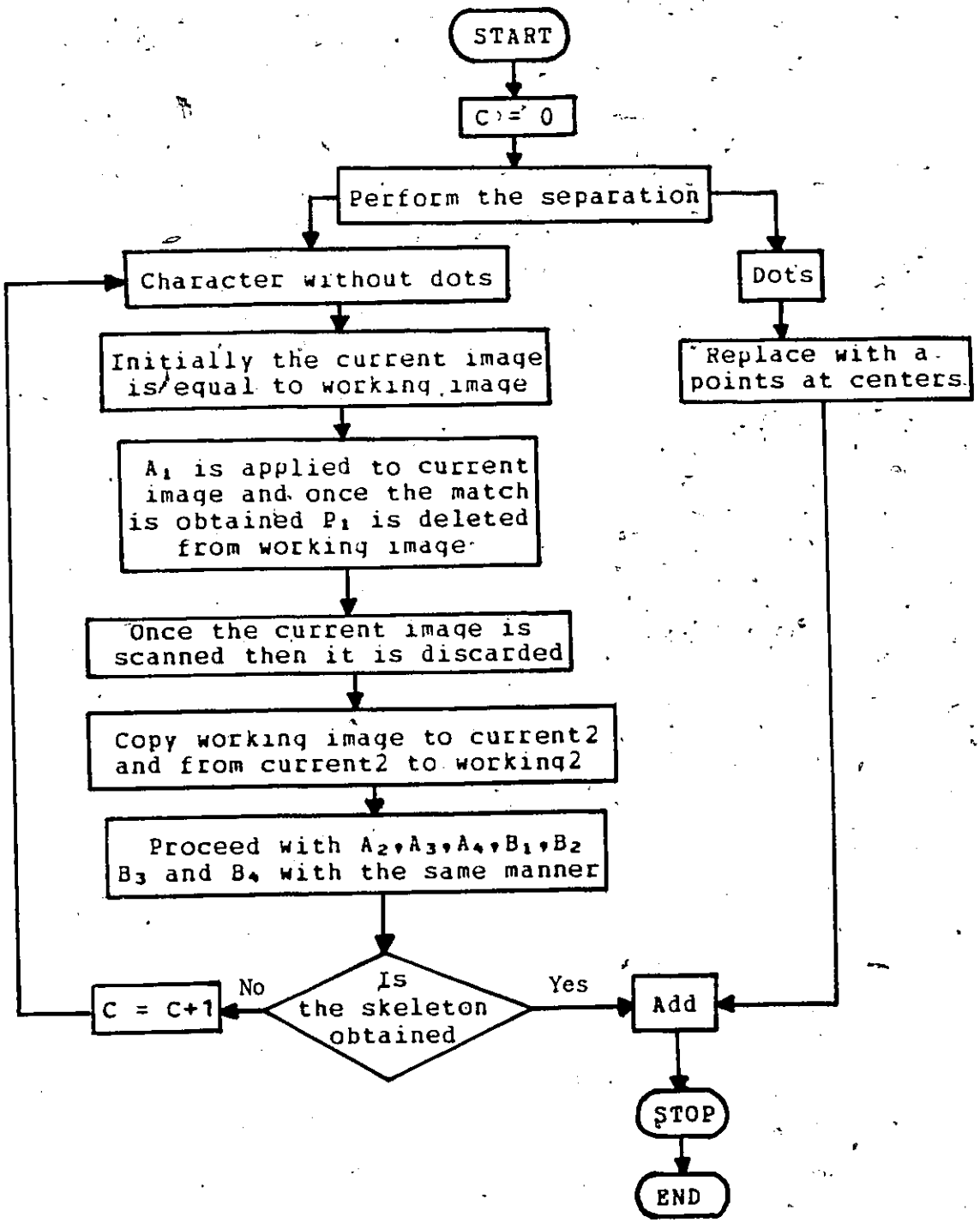


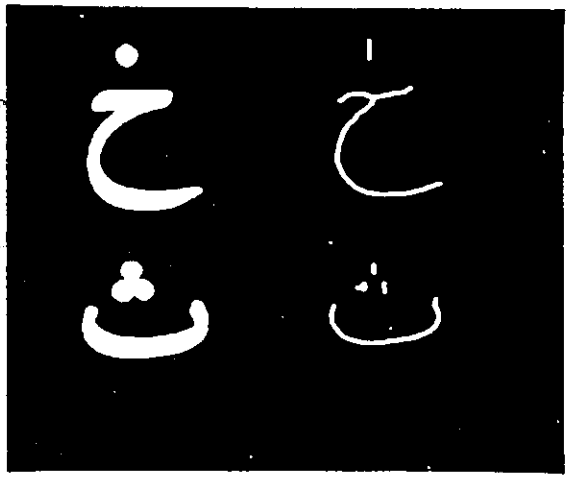
Figure 4.7: Flowchart of the modified approach of matching algorithm

4.3.3 RESULTS

This method works for all Arabic characters and the results are excellent with respect to the preservation of the original image , connectivity and contour noise immunity as shown in Fig.(4.10).

Note that the processing time in this method takes longer than the other methods.

1. Results of using the above algorithm before performing the separation are shown in Fig.4.8
2. An example of separation of the dots from the characters is shown in Fig.(4.9).
3. Results of using the above algorithm after the separation are shown in Fig.(4.10).



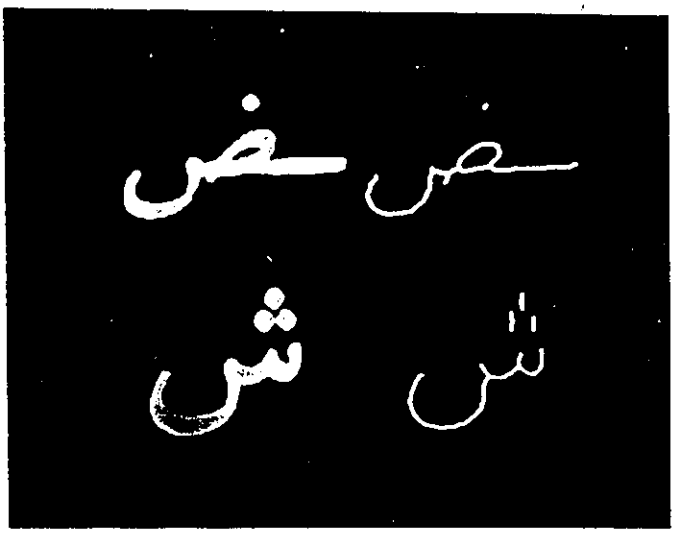


Figure_4.8: Skeletons obtained using the modified approach of the matching algorithm before the separation .

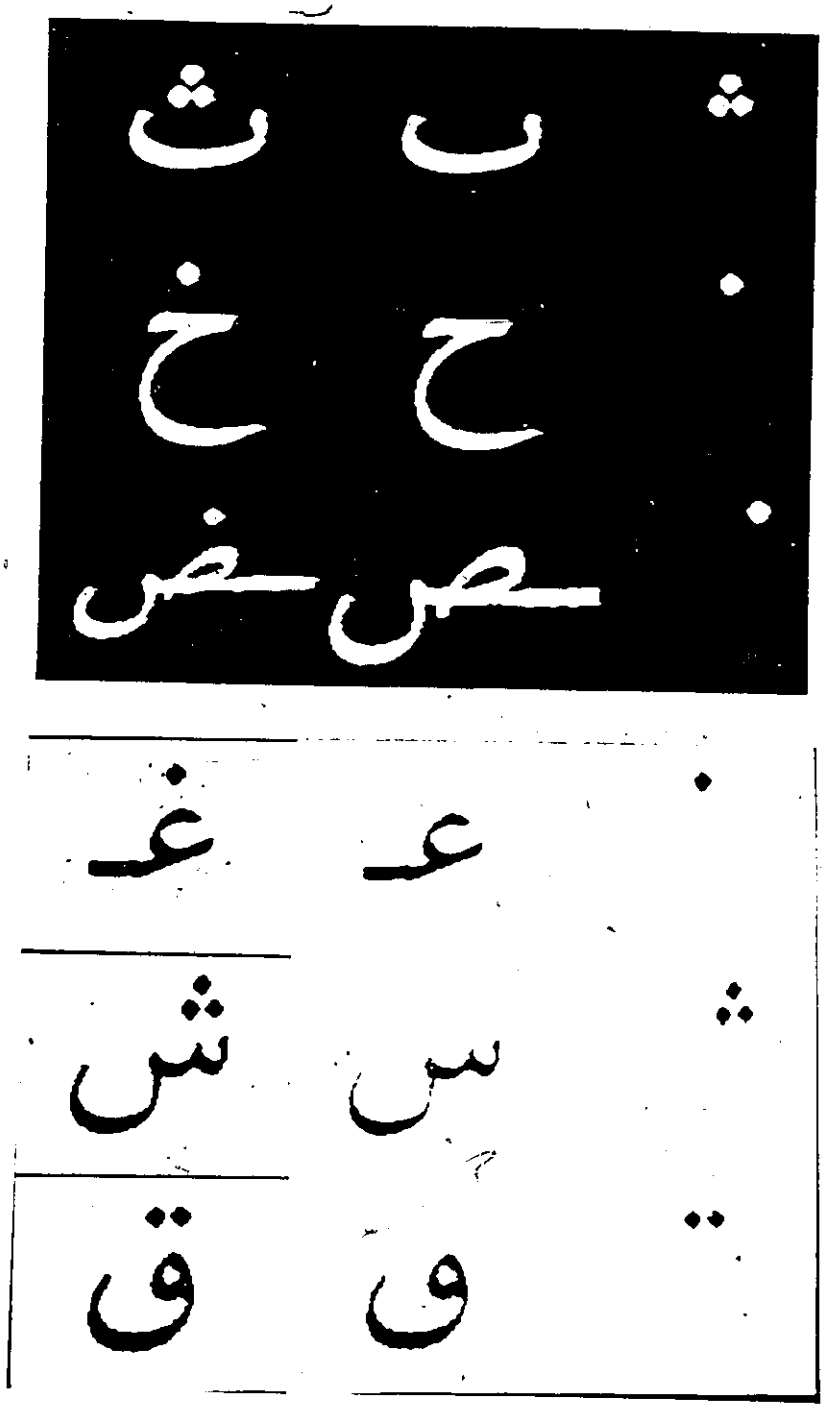
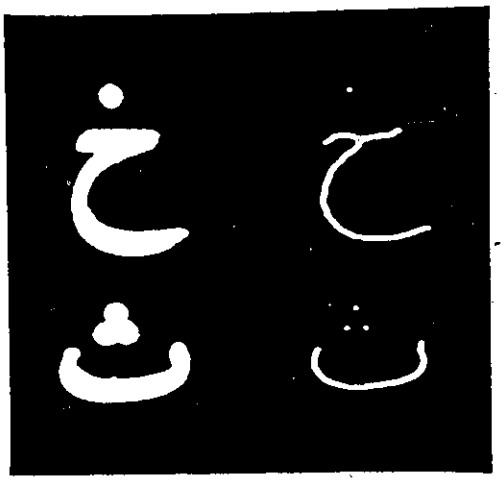


Figure 3.2: An example of separation of the dots from the characters



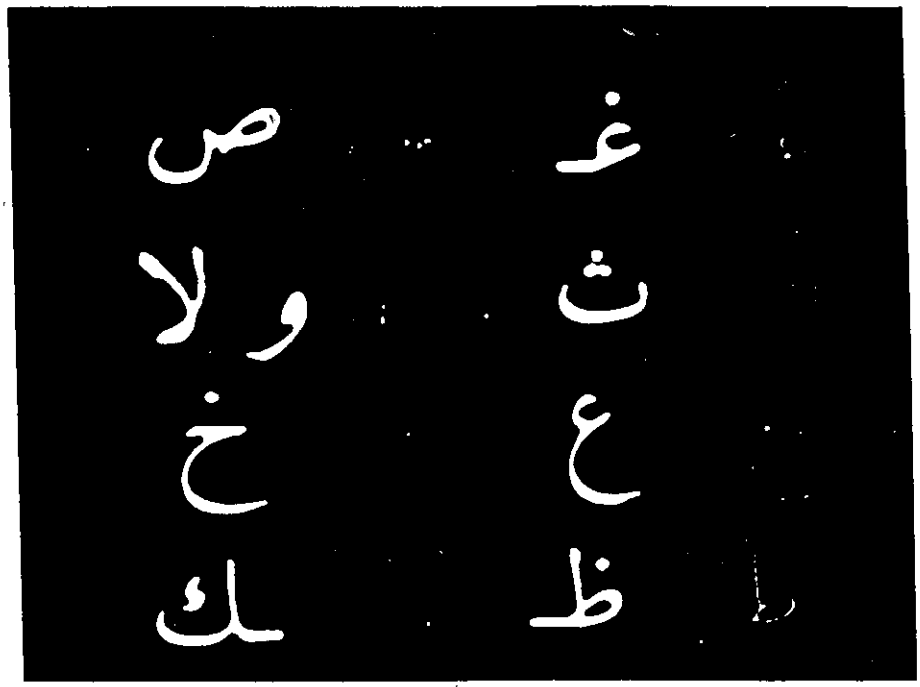


Figure 4.10: Skeletons obtained using the modified approach of the matching algorithm after the separation.

4.4 AMELIORATION OF THE SEQUENTIAL THINNING ALGORITHM

4.4.1 DESCRIPTION

The sequential thinning algorithm presented in Chapter-III- has the problem of disconnectivity.

To minimize this disadvantage we inserted in the program the two first conditions given in the new parallel thinning (see section 4.2).

1. $2 \leqslant B(P_1) \leqslant 6$

2. $A(P_1) = 1$

where $B(P_1)$ is the number of nonzero neighbors of P_1 and $A(P_1)$ is the 01 pattern in the ordered $P_2, P_3, P_4, P_5, P_6, P_7, P_8, P_9$ and P_2 .

Note that condition (1) verifies that P_1 is not an end-point and condition (2) verifies that P_1 is not a break-point (i.e it is not a point which breaks the connectedness of the stroke). Note that P_1 is a candidate for deletion.

By inserting the two conditions the

filename of the program is : DMOD.for

4.4.2 RESULTS

By inserting the two equations (1) and (2) the problem of discontinuity in the skeleton obtained using the sequential thinning algorithm [26-27] can be minimized as shown in Fig.(4.11).

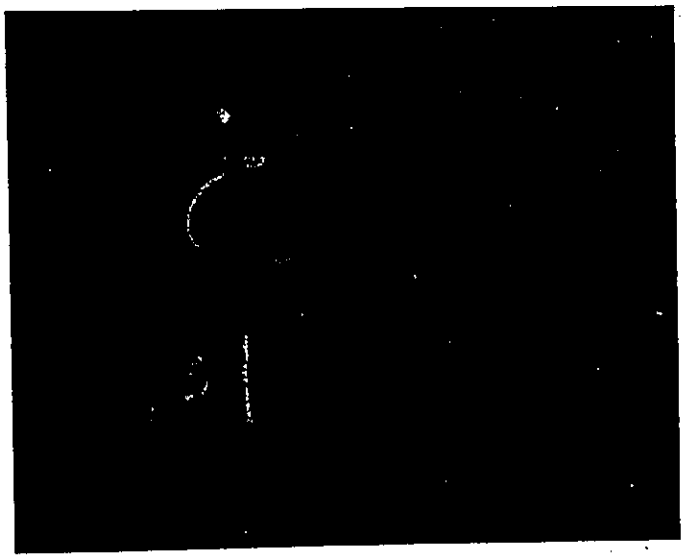


Figure 4.11: Skeletons obtained using the amelioration of the sequential thinning algorithm.

4.5 GENERAL COMPARISON

4.5.1 SIMPLIFIED VERSION OF HILDITCH AND THE NEW PARALLEL THINNING ALGORITHM

The simplified version of Hilditch which is presented by A.Rosenfeld and R.Stafenalli as a formal algorithm may not retain the shape properties of the original shape .

By using the new parallel thinning algorithm which is based on the four sub-iterations we obtain excellent results with respect to both connectivity , contour noise immunity and shape of the original pattern as shown in Fig.(4.12).

4.5.2 MATCHING ALGORITHM AND ITS MODIFIED APPROACH

By using the eight templates (Arcelli's mask) given above and only one image in the processing we obtain a skeleton but not in the middle and with branches in many parts of the character.

By using the same templates and two images (the working image and the current image) we obtain excellent results but only for the character without dots ; for the character with dots we still have branches on the dots.

After separating the dots from the character and processing each part alone as we explained before we obtain excellent results for all Arabic characters with respect to the preservation of the shape properties of the original image , connectivity and contour noise immunity as shown in Fig.(4.13).

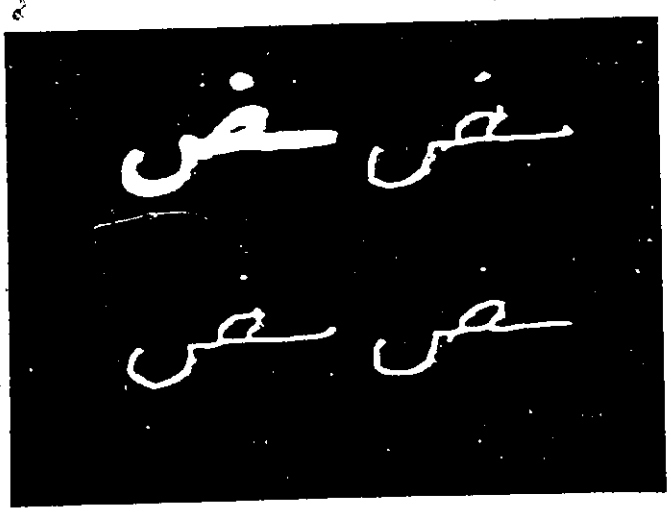
4.5.3 SEQUENTIAL THINNING ALGORITHM AND ITS AMELIORATION

The sequential algorithm which consists of distance transform , new selection rules , linking algorithm and thinning algorithm is the best algorithm concerning the processing time and conserves very well the shape of the original image but has the problem of discontinuity . By inserting the two conditions as it was explained above the problem of discontinuity is minimized as shown in Fig.(4.14).

A general comparison between the methods reviewed and the methods developed is given in the following table.

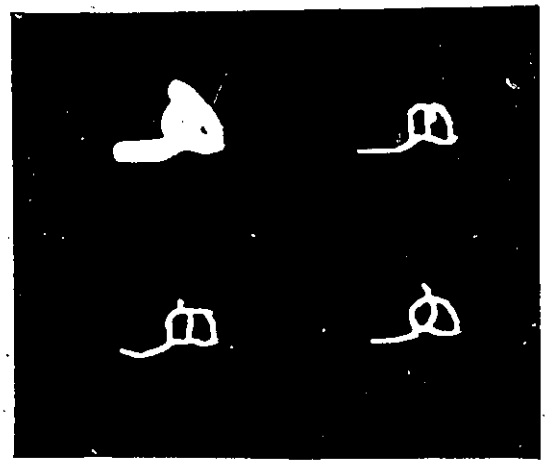
TABLE -1-

ALGORITHMS	ADVANTAGES	DISADVANTAGES
Matching algorithm	good connectivity	Presence of branches
Naccache algorithm	the shape is very well preserved	disconnectivity and branches
Simplified version of Hilditch	good connectivity	the shape properties are not preserved
Sequential thinning algorithm	excellent concerning the processing time	disconnectivity
Modified approach of the matching algorithm	excellent result w.r.t the shape, connectivity and noise immunity.	the processing time is long
New parallel thinning algorithm	same advataqes as in the modified approach	no disdvantaqes



a b

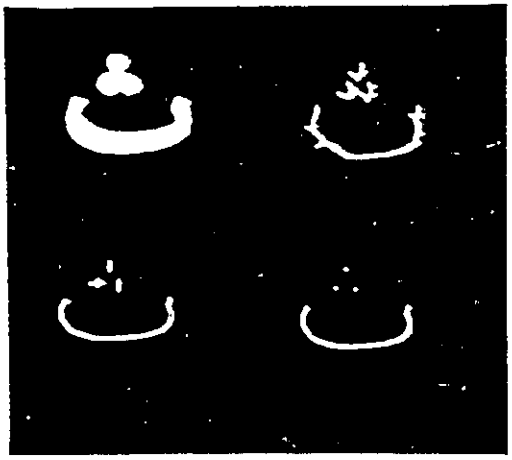
c d



e f

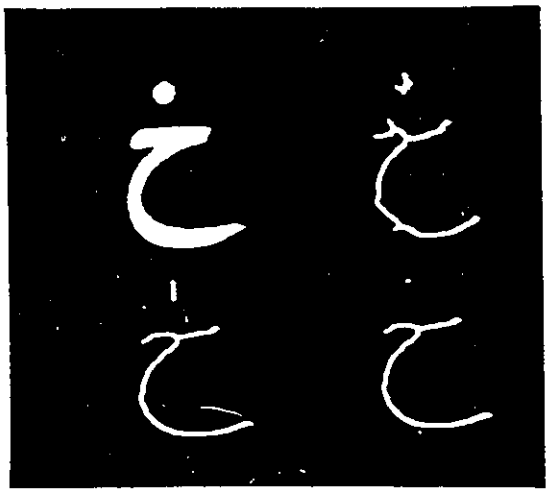
g h

Figure 4.12: (a,e)- Original images ; (b,f)- Skeletons obtained using the simplified version of Hilditch when P_1 is deleted ; (c,g)- Skeletons obtained using the simplified version of Hilditch when P_1 is flagged ; (d,h)- Skeletons obtained using the new parallel thinning algorithm.



a b

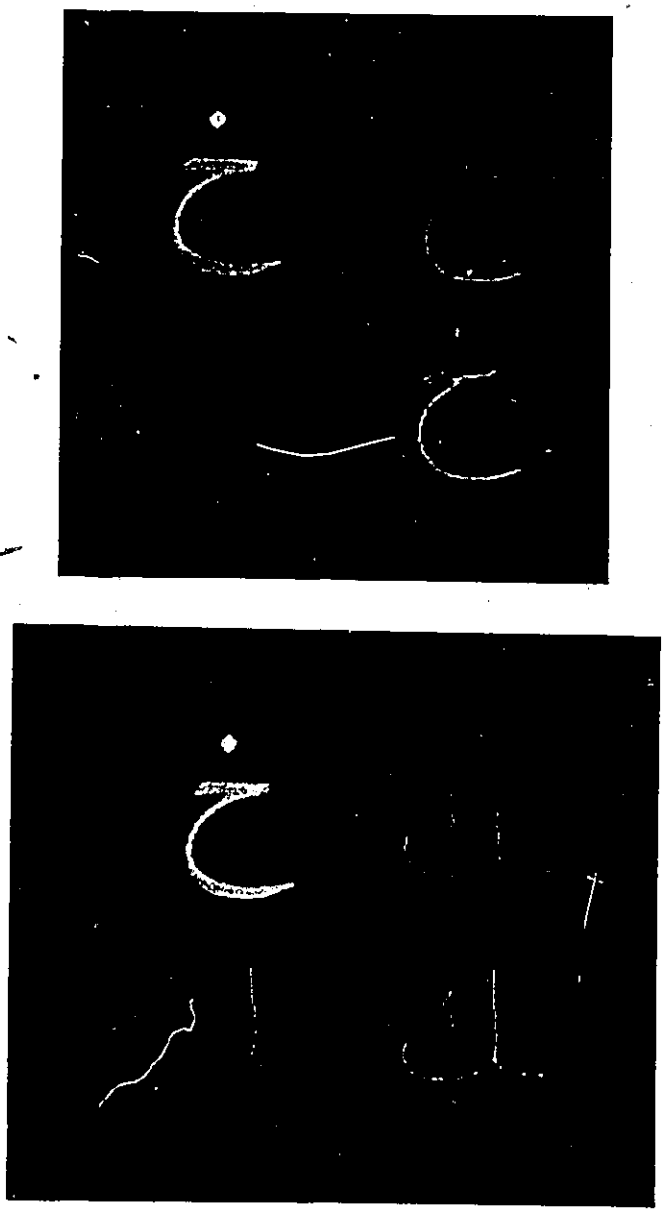
c d



e f

g h

Figure 4.13: (a,e)- Original images ; (b,f)- Skeletons obtained using the eight templates and only one image in the processing [30] ; (c,g)- Skeletons obtained using eight templates and two images (the current image and the working image) in the processing ; (d,h)- Skeletons obtained using (c) and with separation of the dots from the character.



a b

c d

e f

q h

Figure 4.14: (a,e)- Original binary images ; (b,f)- Skeletons obtained using the sequential thinning algorithm with the first equation of new selection rules ; (c,q)- By using the second equation of new selection rules ; (d,h)- Skeletons obtained after inserting the two conditions.

4.6 THICKENING ALGORITHM:

4.6.1 Introduction:

In the following we present a thickening algorithm which is the opposite of the thinning.

4.6.2 Algorithm:

The algorithm consists of iteratively executing many passes over the pattern, where in each pass background points which lie on the boundary are changed to object points.

For the first pass we scan the image row-wise from the upper left corner to the bottom right corner and for every change from 0 to 1 (i.e from the background to the object pixel) or 1 to 0 (i.e from the object pixel to the background pixel) we label the background with the value (2) which means that the value (2) will belong to the object after the second pass.

For the second pass we scan the image column-wise from the upper left corner to the bottom right corner and also for every change from 0 to 1 or 1 to 0 we label the background by the value 2 .

Once the second pass is terminated all the pixels with label (2) are then changed to the value 1 which is the value of the object.

The process is repeated in the same manner as explained above until the desired thickness is obtained.

the filename of the program is : thick.for

The flowchart of thickening algorithm is shown in Fig.(4.15).

4.6.3 RESULTS

Using this method for the thickening algorithm we obtain excellent results .

Results of using the above algorithm by taking as input the original image are shown in Fig.(4.15).

Results of using the same algorithm but taking the thinned characters as the original image are shown Fig.(4.17).

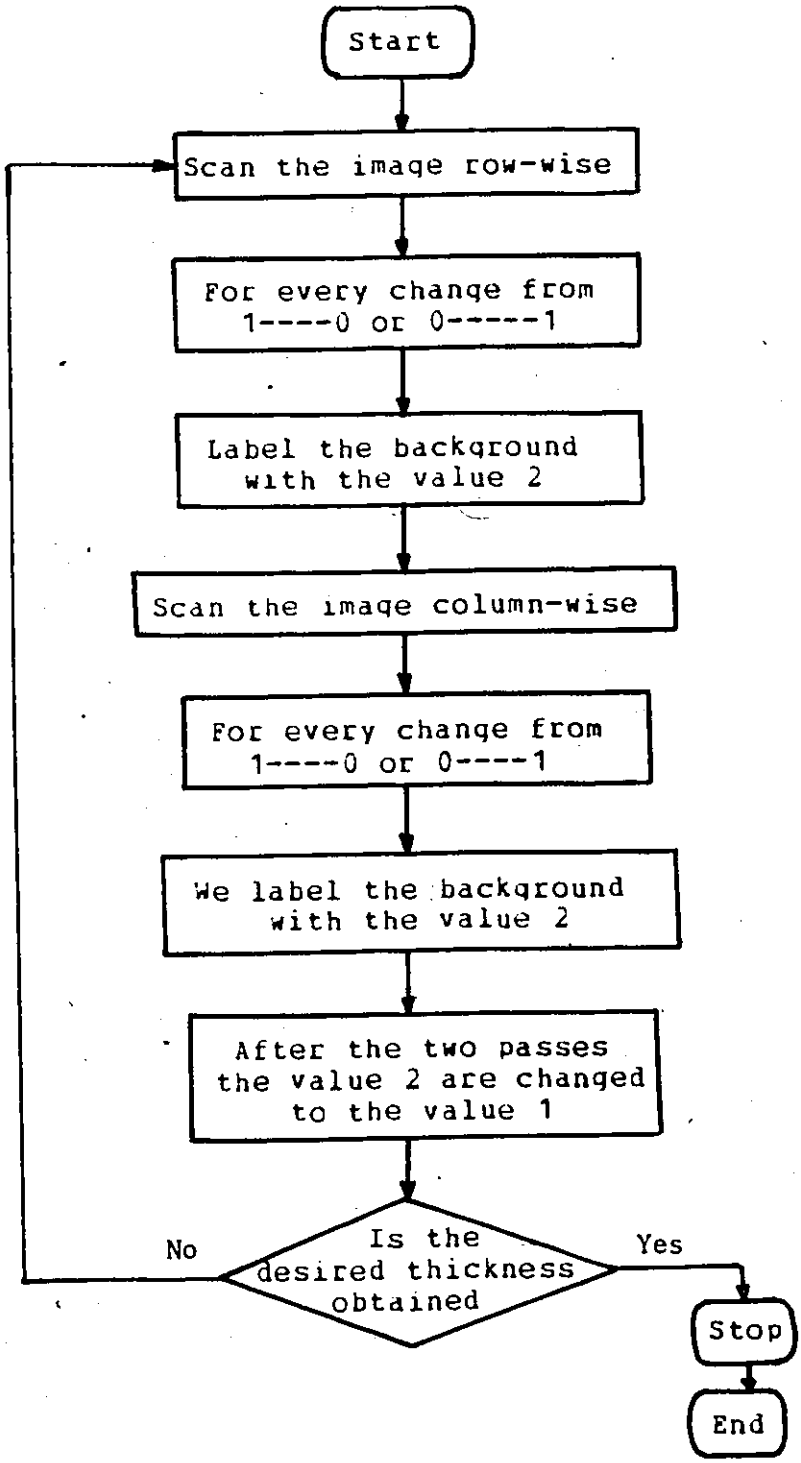


Figure 4.15: Flowchart of the thickening algorithm

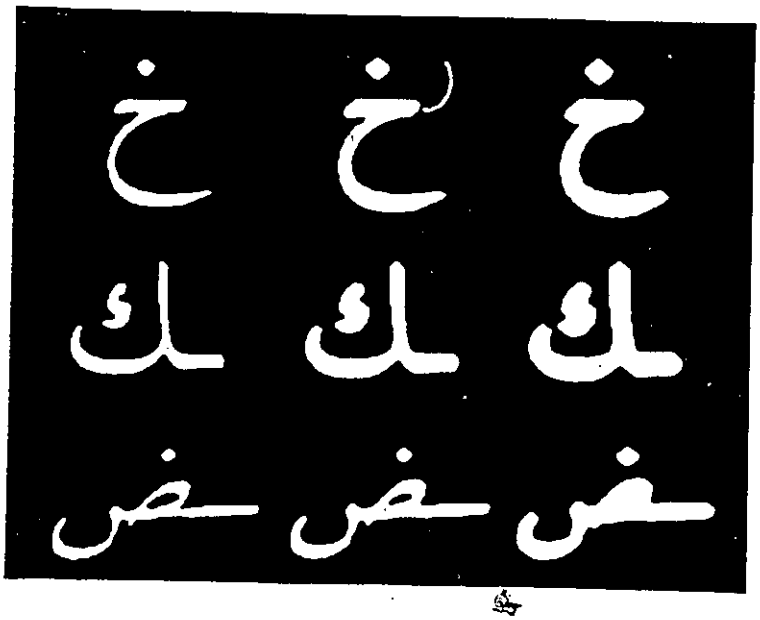


Figure 4.16: Results of using the above algorithm with thickness 2 and 4.

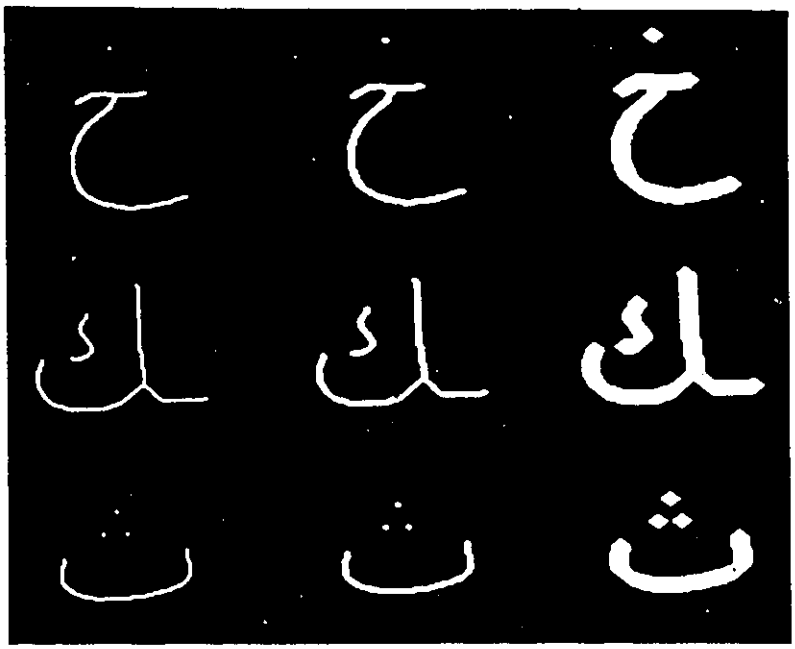


Figure 4.17: Results of using the above algorithm with thickness 2 and 4

4.7 CONCLUSION:

We have shown that the new parallel thinning algorithm and the modified approach of the matching algorithm preserve very well the shape of the original image and yield excellent results with respect to both connectivity and contour noise immunity when applied to Arabic fonts. We have also experimentally compared our algorithms with the reviewed algorithms presented in Chapter-III-, finally a thickening algorithm is presented.

Note that the new parallel thinning algorithm is better than the modified approach of the matching algorithm concerning the processing time. Both algorithms yield excellent results with respect to the connectivity, shape of the original image and contour noise immunity.

Chapter V

SUMMARY AND CONCLUSIONS :

In this thesis we have presented various thinning algorithms although we had discussed all the conditions necessary before a dark point could be deleted from a pattern.

The parallel thinning algorithm presented by Naccache in 1984 preserves very well the shape of the original image but has two problems : 1) existence of extraneous pixels (branches); 2) the final skeleton is not connected.

For the matching method using the eight templates given by Arcelli [30] we have shown that if only one image is used in the processing, we obtain a skeleton but not in the middle and with branches in many parts of the character , and by using two images (the working image and the current image) we obtain excellent results for the characters without dots , but for those with dots we still have branches on the dots . To remove this problem we separate the dots from the character and we treat each part separately. Using this method we obtain excellent results with repect to connectivity, shape of the original image and contour noise immunity.

We have shown that the "simplified version" of the thinning algorithm presented by Stefanelli and Rosenfeld [23] may not retain the shape properties of the input pattern and presence of branches in the skeleton either by flagging the pixels in each pass and deleting the flagged pixels after the last iteration or deleting a pixel each time the four conditions are satisfied. By using the new parallel thinning algorithm which consists of the four sub-iterations (each one is divided into four conditions) we obtain excellent results with respect to the shape of the original image , connectivity and contour noise immunity.

The Lee's method which is the best one concerning the processing time and preserves very well the shape of the original image has the problem of discontinuity which is minimized using the two conditions as we explained above.

At the end of the thesis we have presented a thickening algorithm . Note that all these algorithms are applied to "camera captured" Arabic fonts.

A Thinning algorithm was developed in this thesis. It is an excellent reference which can be used in all the applications mentioned in chapter I.

Chapter VI

SOME_OTHER_EXAMPLES

In this section algorithms presented in Chapter III and Chapter IV are applied to hand-written and printed English fonts and some numbers.

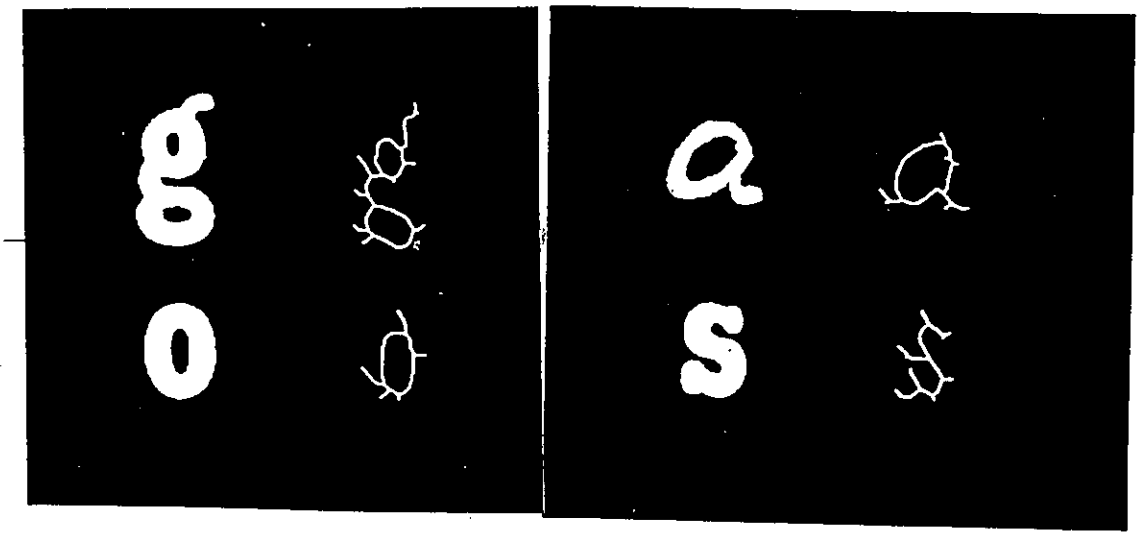


Figure_6.1: Skeletons obtained using the matching algorithm [30]

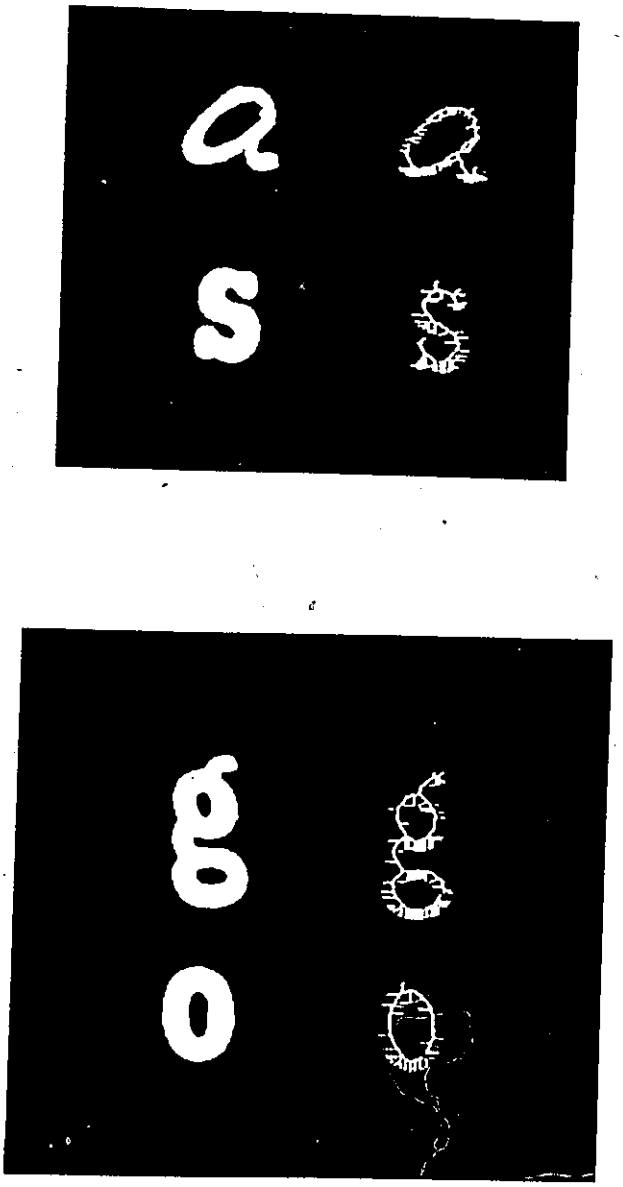
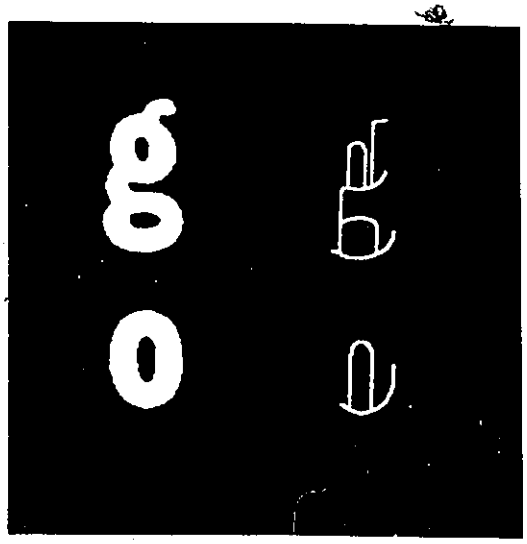


Figure 6.2: Skeletons obtained using Naccache's algorithm [35].



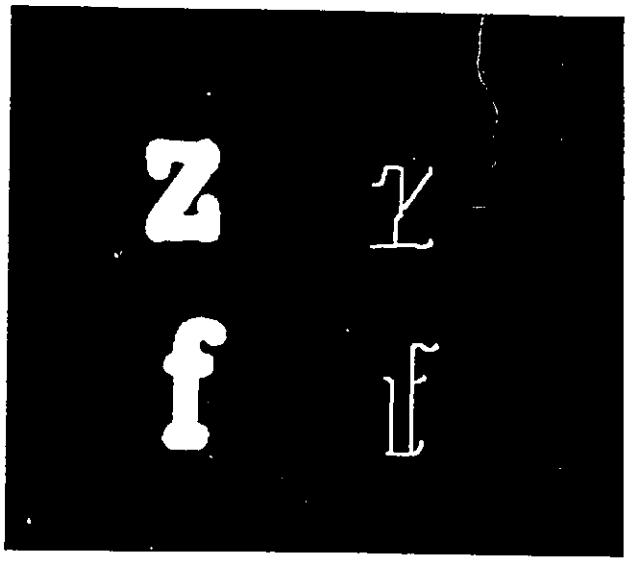
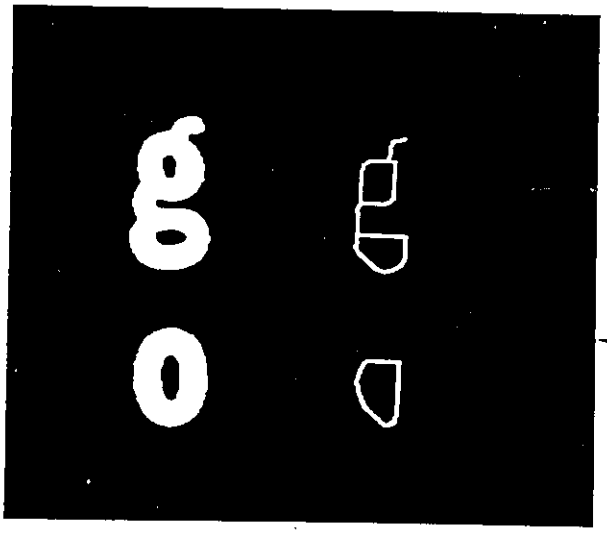


Figure 6.3: Skeletons obtained using the " Simplified version of Hilditch when the central point P_1 is deleted.



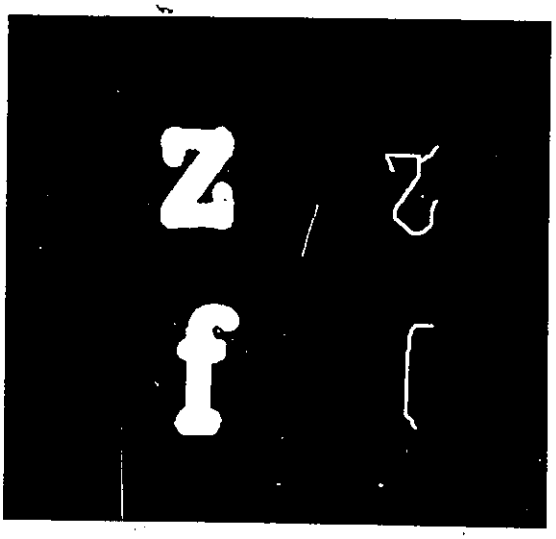


Figure 6.4: Skeletons obtained using the " Simplified version of Hilditch when the central point P_1 is flagged.

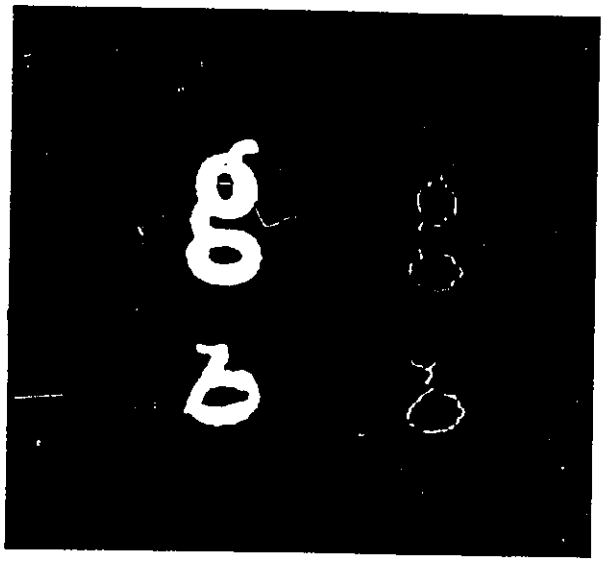


Figure_6.5: Skeletons obtained using the sequential thinning algorithm by using the first equation of the new selection rules.

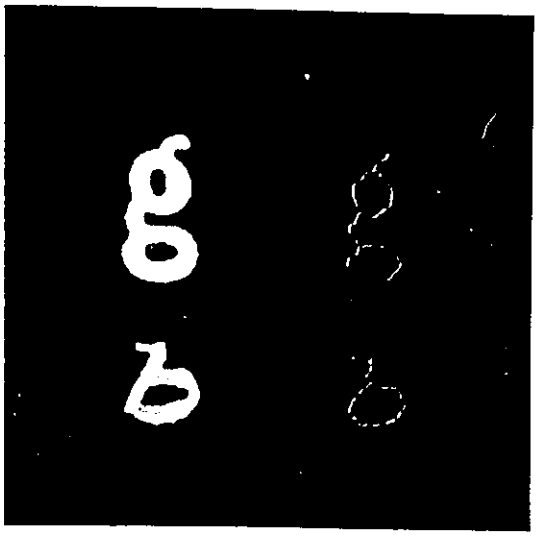


Figure_6.6: Skeletons obtained using the sequential thinning algorithm by using the second equation of the new selection rules.

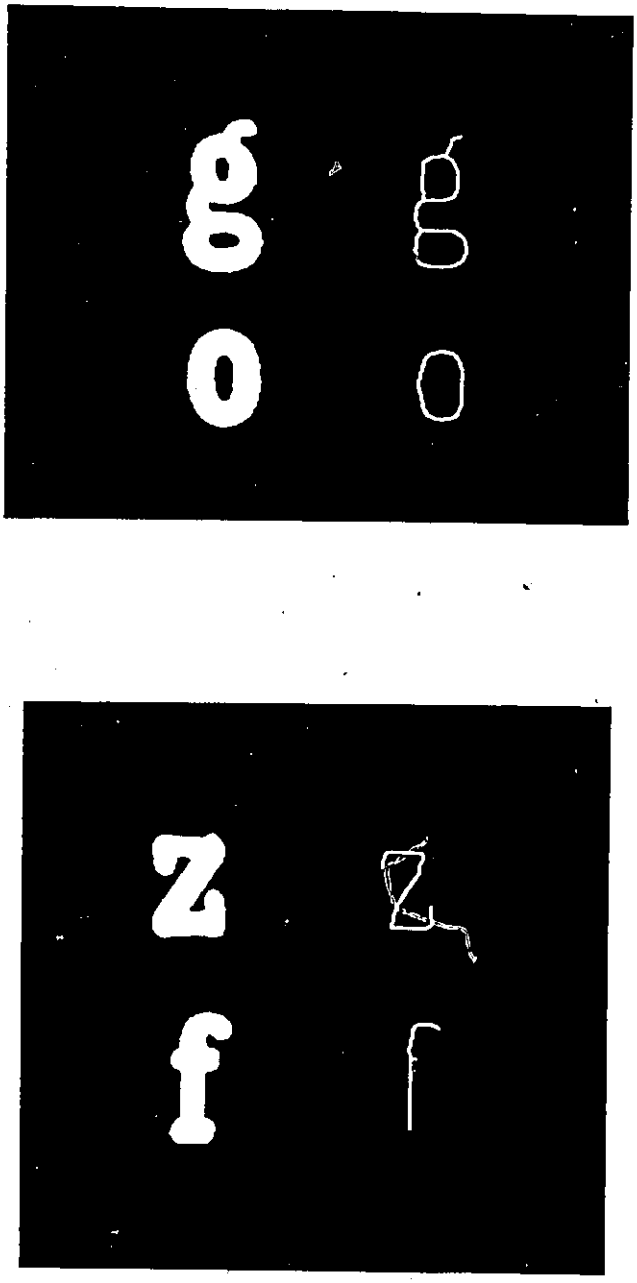
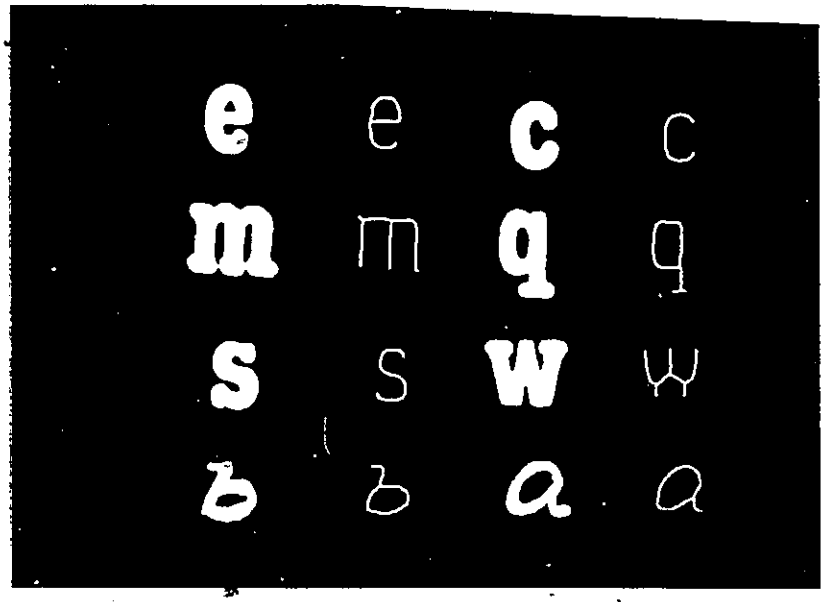


Figure_6.7: Skeletons obtained using the new parallel thinning algorithm



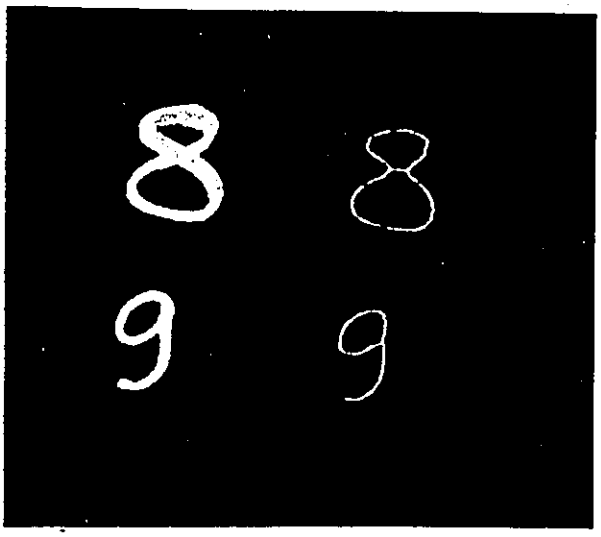


Figure 6.8: Skeletons obtained using the new parallel thinning algorithm

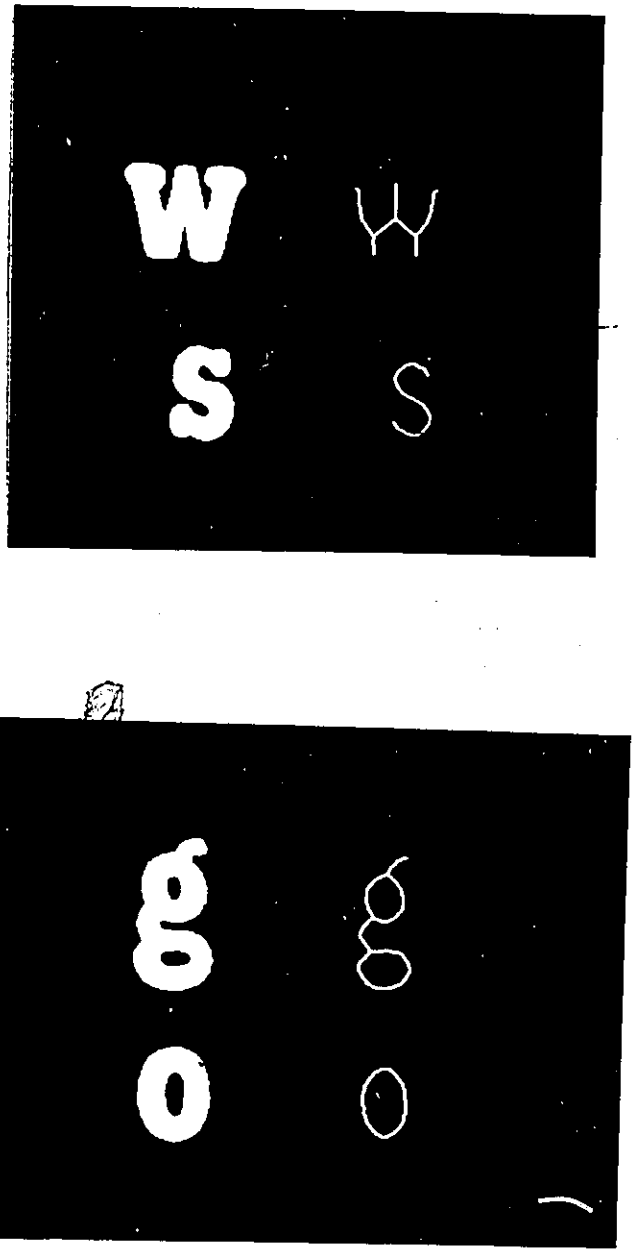


Figure_6.9: Skeletons obtained using the modified approach of the matching algorithm

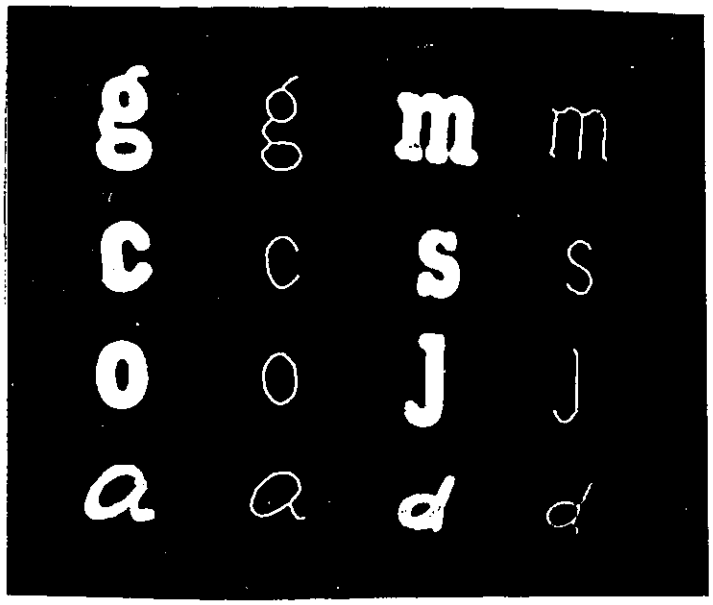


Figure_6.11: Skeletons obtained using the modified approach of the matching algorithm

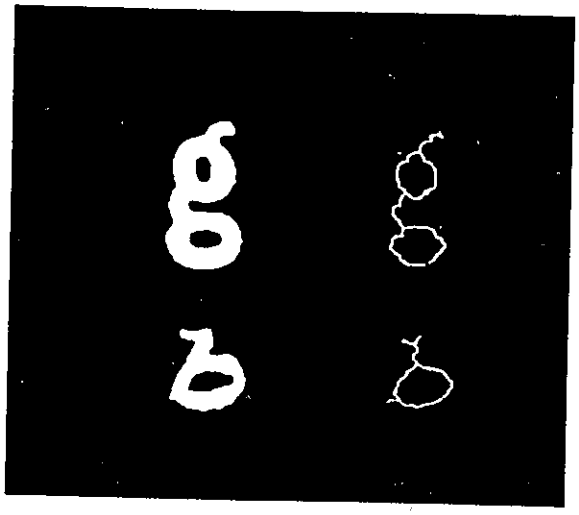
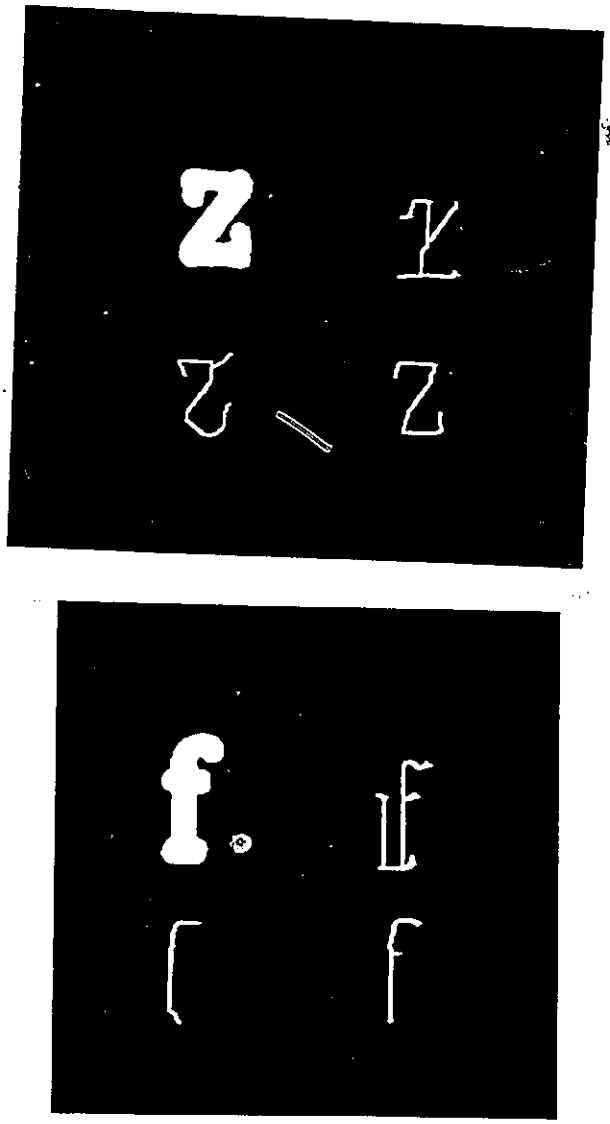


Figure_6.12: Skeletons obtained using the amelioration of the sequential thinning algorithm.



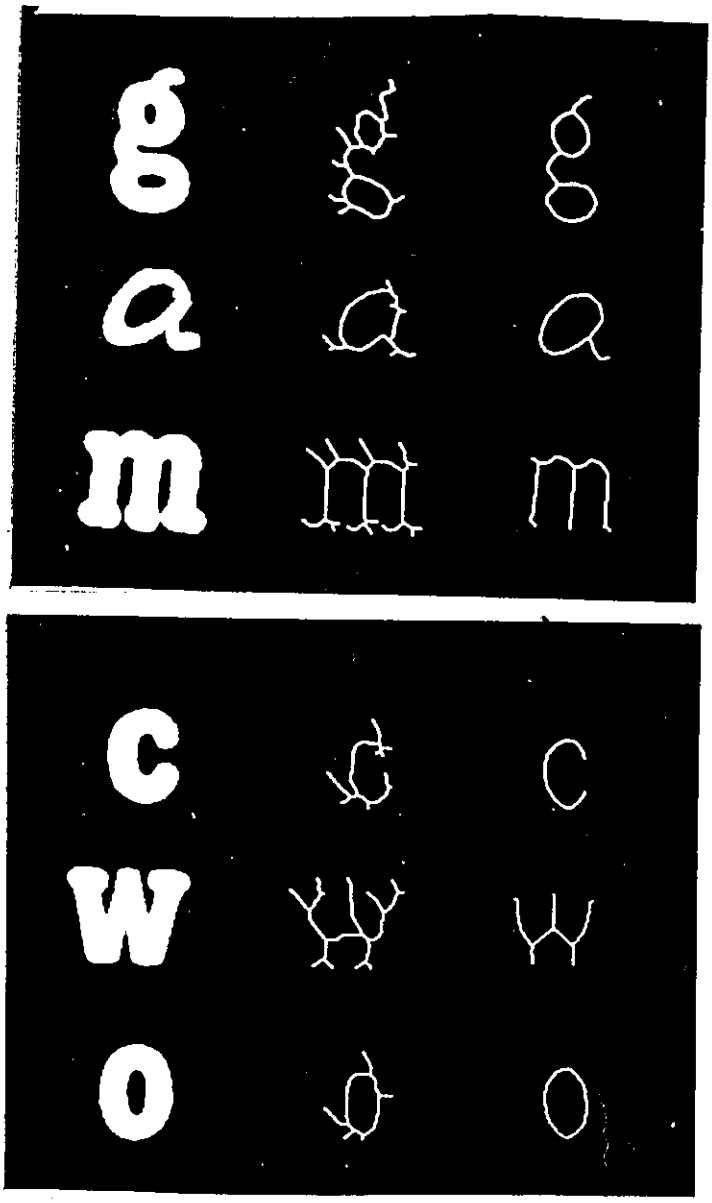
a b

c d

e f

q h

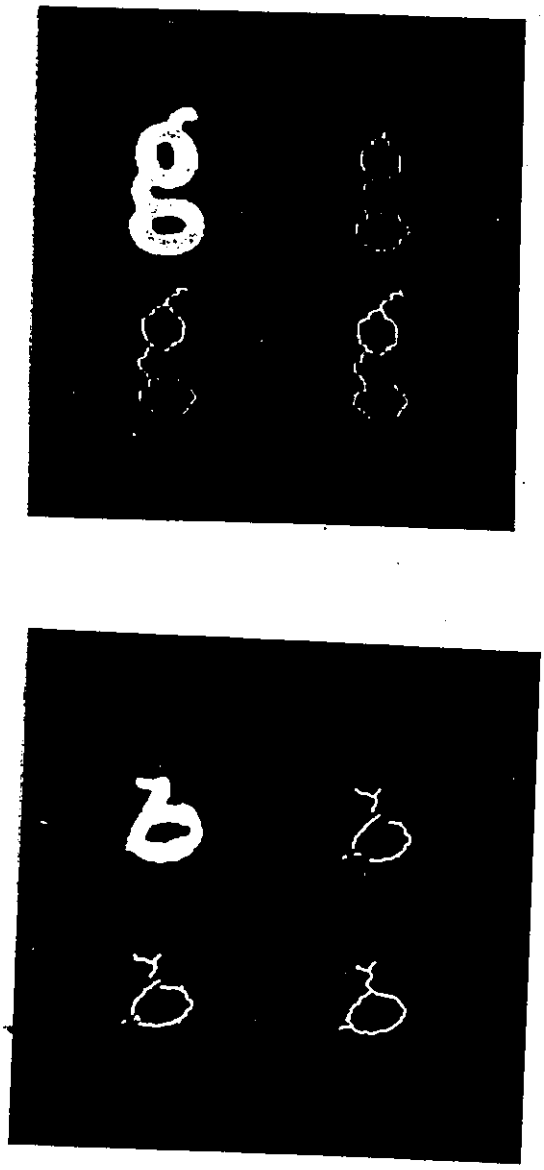
Figure 6.13: (a,e)- Original images ; (b,f)- Skeletons obtained using the simplified version of Hilditch when P_1 is deleted ; (c,q)- Skeletons obtained using the simplified version of Hilditch when P_1 is flagged ; (d,h)- Skeletons obtained using the new parallel thinning algorithm.



a b c
d e f
g h i

a b c
d e f
g h i

Figure 6.14: (a,d,g)- Original images ; (b,e,h)- Skeletons obtained using the eight templates and only one image in the processing [30] ; (c,f,i) Skeletons obtained using eight templates and two images (the current image and the working image) in the processing ;



a b

c d

e f

q h

Figure 6.15: (a,e)- Original binary images ; (b,f)- Skeletons obtained using the sequential thinning algorithm by using the first equation of new selection rules ; (c,q)- By using the second equation of new selection rules ; (d,h)- Skeletons obtained after inserting the two conditions.

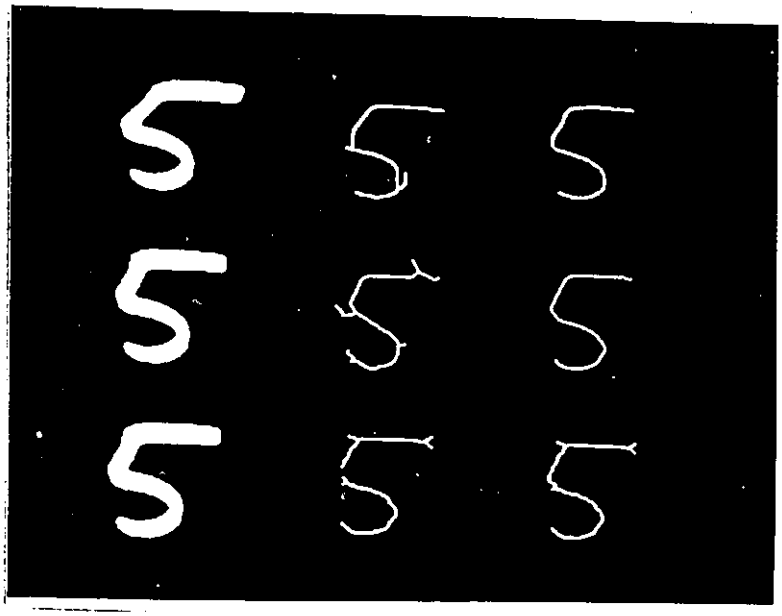


Figure 6.10: General comparison by using numbers

REFERENCES

1. R.C. Gonzalez, and P. Wintz, "Digital Image Processing", Addition Wesley Publishing Company, Inc, 1977.

2. T. Pavilidis, "Computer Graphic and Image Processing", Vol.13, pp.142-157, 1980.

3. W. Pratt, "Digital Image Processing" Vol.1, pp.514-520, 1978.

4. J. Serra, "Image Analysis and Mathematical Morphology", Academic Press Inc (London) LTD, 1982.

5. U. Montarani, "Continuous Skeletons from Digitized Image", J.Ass. Comput. Mach.16, pp.534-549, 1969.

6. Peleg and A. Rosenfeld, "A Min-Max Medial Axis Transformation", IEEE Trans on Pattern Anal.Mach in Intell.3, pp.208-210, 1981.

7. J.L. Pfaltz and A. Rosenfeld, "Computer Representation of Planar Regions by their Skeletons", C.ASS. Comput. Mach.10, pp.119-125, 1967.

8. E.R. Davies and A.P.N. Plummer, "Thinning Algorithms : A Critique and New Methodology", Pattern Recognition.14, pp.53-63, 1981.

9. F.L. Alt, "Digital Pattern Recognition by Moments", J.ACM 9 and 2, pp.240-258, April 1962.

10. K.M.Hu, "Visual Pattern Recognition by Moment Invariants", IRE Trans IT8, pp.179-187, Feb 1962.

11. A. Shimrel, "A Logical Program for the Simulation of Visual Pattern Recognition", In Principles of Self Organization, H.V. Foerster and G.W. Zopf, Eds Perqamm Press, New York, pp.521-526, 1962.

12. C.J. Hilditch, "Linear Skeletons from Square Cupboard", In Machine Intelligence, Vol.4, B.Meltzer and D.Michie, Eds Edinburgh University Press, pp.403-420, 1969.

13. E.S. Deutsch, "Thinning Algorithm on Rectangular, Hexagonal and Triangular arrays", Communication. ACM, Vol.15, pp.827-837, 1972.

14. R.M. Smith, "Computer Processing of Line Images : A Survey", Pattern Recognition, Vol.20 No.1, pp.7-15, 1987.

15. T. Wakayama, "A Core Line Tracing Algorithm Based on Maximal Square Moving", IEEE Transactions on Pattern Recognition and Machine Intelligence, Vol Pami.3, pp.134-143, 1981.

16. T.M. Alcorn and C.W. Hoggar, "Pre-processing of Data for Character Recognition", Martoni Rev.32, pp.61-81, 1969.

17. P. Saraga and D.J. Woollons, "The Design of Operators for Pattern Processing", Proc. IEEE Conf. on Pattern Recognition CP.42, pp.106-116, July 1968.

18. G.P. Dinneen, "Programming Pattern Recognition", Western Jt. Comput. Conf, pp.94-100, 1955.

19. S.H. Unger, "Pattern Detection and Recognition", IRE Proc.47, pp.1737-1752, 1959.

20. H. Sherman, "A Quasi-Topological Method for the Recognition of Line Pattern", Information Processing Proc.UNESCO Conf. Paris, pp.232-238, Butterworths, London, 1959.

21. M. Beum, "A Flexible Method for Automatic Reading of Handwritten Numericals", Philips Tech. Rev, pp.89-101,130-137, 1973.

22. K.J. Udupa and I.S.N. Murphy, "Some New Concepts for Encoding Line Pattern", Pattern Recognition No.7, pp.225-233, 1975.

23. R. Stefanelli and A. Rosenfeld, "Some Parallel Thinning Algorithms for Digital Picture", J. ACM 18, pp.255-264, April 1971.

24. A. Rosenfeld and L.S. Davis, "A Note on Thinning", IEEE Transaction on Systems, Man and Cybernetics, pp.236-238, March 1976.

25. H. Tamura, "A Comparaison of Line Thinning Algorithms from Digital Geometry viewpoint", Proc. 4th. Jt Conf on Pattern Recognition, Kyoto, Japan, pp.715-719, 1978.

26. C.C. Lee, Modified_Distance_Transform_and_Linking Algorithm_for_Image_Skeletonization_", Spie Proceedings, Vol.415, Coherent Infrared Radar Systems and Application(II), Arlington, pp.4-8, April 1983.

27. C.C. Lee, "_A_Sequential_Thinning_Algorithm_for_Image Skeletonization_", Spie Proceeding, Vol.415, Coherent Infrared Radar Systems and and Application(II), Arlington, 1984.

28. J.T. Kuehn, J.A. Fessler, and H.J. Siegel, "_Parallel Image_Thinning_and_Vectorization_on_PASM_", Proc. IEEE, pp.368-374, 1985.

29. C. Arcelli and G.S.Di-Baja, "_A_Thinning_Algorithm Based_on_Promise_", Pattern Recognition Society, pp.225-235, Nov 1980.

30. C. Arcelli, L. Cordella and S.Levialdi "_Parallel Thinning_of_Binary_Pictures_", Electronics Letters, Vol.11, pp.138-149, July 1975.

31. C.J. Hilditch, "_Comparaison_of_Thinning_Algorithms_on a_Parallel_Processor_", Image and Vision Processing, Vol.1, pp.115-132, Aug 1983.

32. A. Bel-lan and L. Montoto, "_A_Thinning_Transform_for Digital_Images_", Signal Processing No.3, pp.37-47, 1981.

33. C. Arcelli, L.P.Cordella, and S. Levialdi, "_From Local_Maxima_to_Connected_Skeletons_", Vol. PAMI No.2, March 1981.

34. T. Pavilidis, "_Algorithms_for_Graphics_and_Image Processing_", Rockville MD : Computer Science Press, pp.195-214, 1982.

35. N.J. Naccache and R. Shinghal, "_An_Investigation_into Skeletonization_Approach_of_Hilditch_", Pattern Recognition, Vol.17 No.3, pp.279-284, 1984.

36. A.R. Dill, M.D. Levine, and P.B. Noble, "_Multiple Resolution_Skeletons_", IEEE Transaction on Pattern Recognition and Machine Intelligence, Vol PAMI.9 No.4, pp.495-503, July 1987.

37. G. Bertrand, "_Skeletons_in_Derived_Grides_", E.S.I.E.E, pp.326-329, 1985.

38. A. Favre, HJ. Keller, "_Parallel_Syntatic_by_Recoding of_Binary_Pictures_", Computer Vision, Graphic and Image Processing No.23, pp.99-112, 1983.

39. T. Pavilidis, "A Flexible Parallel Thinning Algorithm", Proc. IEEE Comput. Soc. Conf. on Pattern Recognition and Image Processing, pp.162-167, Aug 1981.

40. A. Rosenfeld and J.L. Pfaltz, "Sequential Operations in Digital Picture Processing", J. ACM, Vol.15, pp.471-494, 1966.

41. U. Montarani, "A Method for obtaining Skeletons using a Quasi Euclidean Distance", J. ACM, Vol.15, pp.600-624, 1968.

42. A. Rosenfeld, "Connectivity in Digital Picture", J. ACM, Vol.17 No.1, pp.146-163, 1970.

43. Y.N. Chu and C.Y. Suen, "An Alternate Smoothing and Stripping Algorithm for Thinning Digital Binary Pattern", Signal Processing, Vol.No3, pp.207-222, 1986.

44. N.J. Naccache and R. Shinghal, "SPTA : A Proposed Algorithm for Thinning Binary Pattern", IEEE Transaction on Systems, Man, and Cybernetics, Vol. SMC.14 No.3, pp.409-418, 1984

45. H. Ogawa and K. Taniguchi "Thinning and Stroke Segmentation for Handwritten Chinese Character Recognition", Pattern Recognition, Vol.15 No.4, pp.299-308, 1982.

APPENDIX - A -

GRAY LEVEL THRESHOLDING.

This approach consists of dividing the gray-level scale into bands, and then using thresholds to determine regions or to obtain boundary points. The threshold value, is the gray-level value which separates the object and the background of an image. All pixels with gray-level value below the threshold T, shown in Fig.(1) are declared as '0' and the levels above T as '255' (this is valid for the binary transformation). This technique is called single-level thresholding and T is known as the threshold value of the image. However there might be cases where the image might contain more than two distinct populations, as shown by the graph given in Fig.(2). In these cases, it is then required to group the levels into more than two populations.

Then we have a multi-level thresholding with different threshold values (for the histogram given as an example in Fig.(2), we have two threshold values T_1 and T_2,

For the single level thresholding the objective is to select T (threshold value), such that the band B_1 (see Fig.(1)) will contain as closely as possible, levels associated with the background, while B_2 will contain the levels of the object.

Threshold selection for an image histogram is thesame shape as the one given in Fig.(1) or (2) is quite straightforward. The threshold is selected at the bottom of the valley between two peaks. There exist methods, which consist of transforming the histogram of an image (when the task of selecting a good threshold would be quite difficult) to a shape where threshold selection would not pose a problem, as the ones shown in Figs.(1), and (2). Such techniques have been investigated by several authors for more details refer to [1].

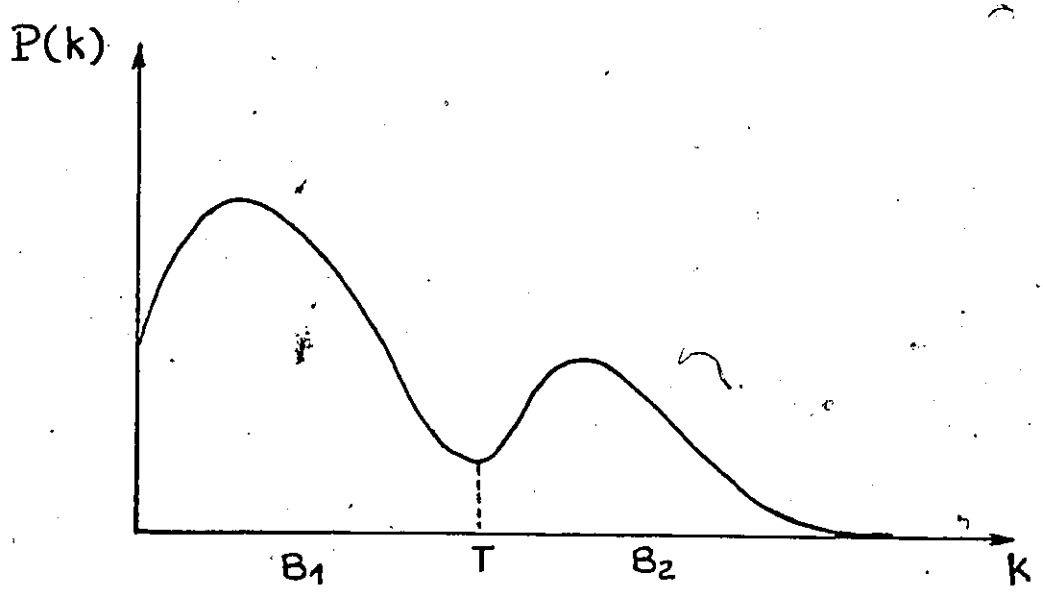


Figure_1: A sample histogram illustrating a bi-modal distribution

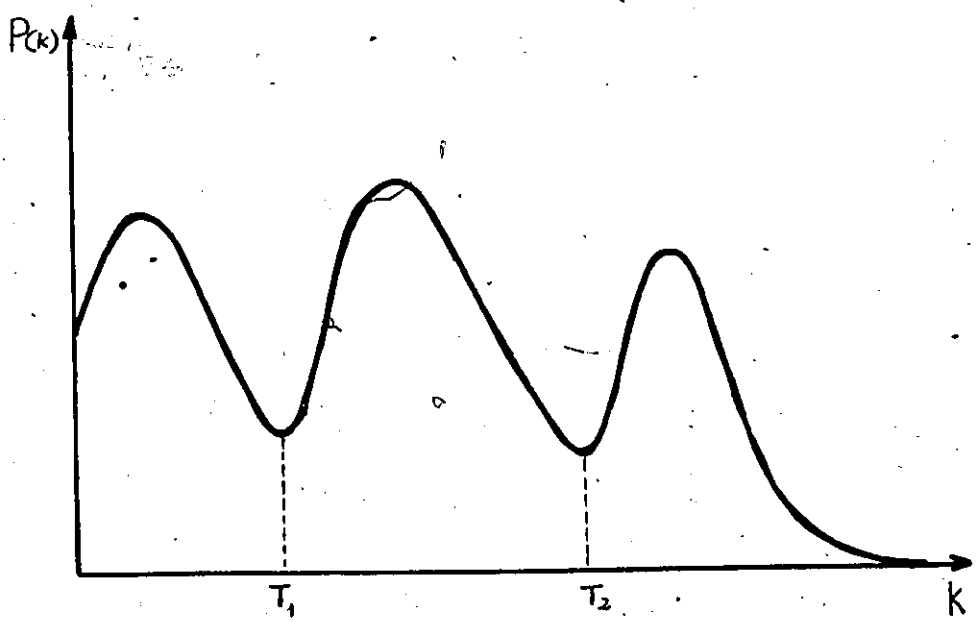


Figure_2: A sample histogram illustrating a multi-modal distribution.

APPENDIX - B -

BORDER FOLLOWING ALGORITHM

TABLE 1

Co-ordinates of eight neighbors

	LX(J)	LY(J)
1.	ID	JD
2.	LX(1)+K1	LY(1)-K2
3.	LX(2)-K2	LY(2)-K1
4.	LX(3)-K2	LY(3)-K1
5.	LX(4)-K1	LY(4)+K2
6.	LX(5)-K1	LY(5)+K2
7.	LX(6)+K2	LY(6)+K1
8.	LX(7)+K2	LY(7)+K1

Co-ordinates of the border element : (I1,J1)

Co-ordinates of the first neighbor : (Id,Jd)

K1 = Jd-J1 K2 = Id-I1

The border following algorithm consists of the following six steps .

1- Detect the first border element at (I1,J1) (a dark pixel) through a row (or column) scan. The element immediately preceding (I1,J1) is labelled as the first neighbor (Id,Jd).

2- Starting with (Id,Jd) and proceeding clock-wise, label the other seven neighbors of (I1,J1) as 2,3,.....,8 , set K=2.

3- Evaluate the co-ordinates LX(K), LY(K) of the K neighbor of (I1,J1) using table 1 .

4- If the pixel at the K is a '1' (i.e a dark point pixel), then this pixel is the next border element. Define

(I1,J1) as this element and (Id,Jd) as the preceding element. Go to step 2.

5- If the pixel at the k neighbor is a '0', set k=k+1 and goto step 3.

6. Proceed until the first border element detected in step 1 is encountered again.

```
c*****************************************************************
c   filename : temp1.for                                          *
c                                                                 *
c  Description : This program is implemented to perform          *
c                thinning algorithm using matching method         *
c                in which eight templates denoted by A1,B1        *
c                A2,B2,A3,B3,A4 and B4 are used . Note that       *
c                only one image is used in the processing         *
c*****************************************************************
      integer h(128,128),p(9),dd,k1,k2,k3,k4,k5,k6
      character img(128,128)
      integer k7,k8,k9,k10,k11,k12,k13,k14,k15,k16
      character*16 filnme,filename
      write(*,*)' enter the input filename '
      read(*,'(a16)') filnme
      write(*,*) 'enter the output filename'
      read(*,'(a16)') filename
      write(*,*)' enter the size of the image ---->'
      read(*,*) nsize
      write(*,*)' enter the scaling factors over X and Y '
      read(*,*) scalx,scaly
      write(*,*)' 1'
      open(1,file=filnme,form='binary',status='old')
      open(2,file=filename,form='binary',status='new')
      write(*,*)' 2'
      do 20 i=1,nsize
      do 20 j=1,nsize
      read(1) img(i,j)
      h(i,j)=ichar(img(i,j))
      if(h(i,j).eq.0)then
      h(i,j)=1
      else
      h(i,j)=0
      endif
 20   continue
      write(*,*) 'enter dd'
      read(*,*) dd
      do 111 kk=1,dd
c templates A1 and B1
      do 30 i=1,nsize
```

```
      do 30 j=1,nsize
      if(h(i,j).eq.1) goto 40
      goto 30
40    continue
      p(2)=h(i,j+1)
      p(3)=h(i-1,j+1)
      p(4)=h(i-1,j)
      p(5)=h(i-1,j-1)
      p(6)=h(i,j-1)
      p(7)=h(i+1,j-1)
      p(8)=h(i+1,j)
      p(9)=h(i+1,j+1)
      k1=p(4)+p(5)+p(6)
      k2=p(2)+p(8)
      if(k1.eq.0.and.k2.eq.2) goto 22
      goto 30
22    h(i,j)=0
30    continue
      do 301 i=1,nsize
      do 301 j=1,nsize
      if(h(i,j).eq.1) goto 401
      goto 301
401   continue
      p(2)=h(i,j+1)
      p(3)=h(i-1,j+1)
      p(4)=h(i-1,j)
      p(5)=h(i-1,j-1)
      p(6)=h(i,j-1)
      p(7)=h(i+1,j-1)
      p(8)=h(i+1,j)
      p(9)=h(i+1,j+1)
      k9=p(3)+p(4)+p(5)
      k10=p(7)+p(8)
      if(k9.eq.0.and.k10.eq.2) goto 221
      goto 301
221   h(i,j)=0
301   continue
c templates A2 and B2
      do 12 i=1,nsize
      do 12 j=1,nsize
      if(h(i,j).eq.1) goto 13
```

```
      goto 12
13    p(2)=h(i,j+1)
      p(3)=h(i-1,j+1)
      p(4)=h(i-1,j)
      p(5)=h(i-1,j-1)
      p(6)=h(i,j-1)
      p(7)=h(i+1,j-1)
      p(8)=h(i+1,j)
      p(9)=h(i+1,j+1)
      k3=p(2)+p(3)+p(4)
      k4=p(6)+p(8)
      if(k3.eq.0.and.k4.eq.2) goto 14
      goto 12
14    h(i,j)=0
12    continue
      do 129 i=1,nsize
      do 129 j=1,nsize
      if(h(i,j).eq.1) goto 139
      goto 129
139   p(2)=h(i,j+1)
      p(3)=h(i-1,j+1)
      p(4)=h(i-1,j)
      p(5)=h(i-1,j-1)
      p(6)=h(i,j-1)
      p(7)=h(i+1,j-1)
      p(8)=h(i+1,j)
      p(9)=h(i+1,j+1)
      k11=p(2)+p(3)+p(9)
      k12=p(5)+p(6)
      if(k11.eq.0.and.k12.eq.2) goto 149
      goto 129
149   h(i,j)=0
129   continue
c templates A3 and B3
      do 16 i=1,nsize
      do 16 j=1,nsize
      if(h(i,j).eq.1) goto 17
      goto 16
17    p(2)=h(i,j+1)
      p(3)=h(i-1,j+1)
      p(4)=h(i-1,j)
```

```
      p(5)=h(i-1,j-1)
      p(6)=h(i,j-1)
      p(7)=h(i+1,j-1)
      p(8)=h(i+1,j)
      p(9)=h(i+1,j+1)
      k5=p(2)+p(8)+p(9)
      k6=p(4)+p(6)
      if(k5.eq.0.and.k6.eq.2) goto 18
      goto 16
18    h(i,j)=0
16    continue
      do 161 i=1,nsize
      do 161 j=1,nsize
      if(h(i,j).eq.1) goto 171
      goto 161
171   p(2)=h(i,j+1)
      p(3)=h(i-1,j+1)
      p(4)=h(i-1,j)
      p(5)=h(i-1,j-1)
      p(6)=h(i,j-1)
      p(7)=h(i+1,j-1)
      p(8)=h(i+1,j)
      p(9)=h(i+1,j+1)
      k13=p(7)+p(8)+p(9)
      k14=p(3)+p(4)
      if(k13.eq.0.and.k14.eq.2) goto 181
      goto 161
181   h(i,j)=0
161   continue
c templates A4 and B4
      do 222 i=1,nsize
      do 222 j=1,nsize
      if(h(i,j).eq.1) goto 21
      goto 222
21    p(2)=h(i,j+1)
      p(3)=h(i-1,j+1)
      p(4)=h(i-1,j)
      p(5)=h(i-1,j-1)
      p(6)=h(i,j-1)
      p(7)=h(i+1,j-1)
      p(8)=h(i+1,j)
```

```
      p(9)=h(i+1,j+1)
      k7=p(6)+p(7)+p(8)
      k8=p(2)+p(4)
      if(k7.eq.0.and.k8.eq.2) goto 23
      goto 222
23    h(i,j)=0
222   continue
      do 2221 i=1,nsize
      do 2221 j=1,nsize
      if(h(i,j).eq.1) goto 211
      goto 2221
211   p(2)=h(i,j+1)
      p(3)=h(i-1,j+1)
      p(4)=h(i-1,j)
      p(5)=h(i-1,j-1)
      p(6)=h(i,j-1)
      p(7)=h(i+1,j-1)
      p(8)=h(i+1,j)
      p(9)=h(i+1,j+1)
      k15=p(5)+p(6)+p(7)
      k16=p(2)+p(9)
      if(k15.eq.0.and.k16.eq.2) goto 231
      goto 2221
231   h(i,j)=0
2221  continue
111   continue
      do 131 i=1,nsize
      do 131 j=1,nsize
      if(h(i,j).eq.0) then
      h(i,j)=255
      else
      h(i,j)=0
      endif
131   continue
      do 140 i=1,nsize
      do 140 j=1,nsize
      img(i,j)=char(h(i,j))
140   continue
      write(2) ((img(i,j),j=1,nsize),i=1,nsize)
      close(2)
      stop
      end
```

```
C*****************************************************************
C     filename : alh.for                                         *
C                                                                *
C    Description : This program is implemented to perform       *
C                  thinning algorithm using N.Naccache's        *
C                  method                                       *
C                                                                *
C*****************************************************************
C
      integer h(128,128),b(9),g(9),p(9)
      character img(128,128)
      integer nsize,n,m,sum,xp1,init1,init2,ll,cc
      character*16 filnme,filename
      write(*,*)' enter the input filename '
      read(*,'(a16)') filnme
      write(*,*)'enter the output filename'
      read(*,'(a16)') filename
      write(*,*)' enter the size of the image ---->'
      read(*,*) nsize
      write(*,*)' enter the scaling factors over X and Y '
      read(*,*) scalx,scaly
      write(*,*)' 1'
      open(1,file=filnme,form='binary',status='old')
      open(2,file=filename,form='binary',status='new')
      write(*,*)' 2'
      do 20 i=1,nsize
      do 20 j=1,nsize
      read(1) img(i,j)
      h(i,j)=ichar(img(i,j))
      if(h(i,j).eq.0)then
      h(i,j)=1
      else
      h(i,j)=0
      endif
 20   continue
      close(1)
      write(*,*) 'enter cc'
      read(*,*) cc
      do 7 kk=1,cc
      do 30 i=1,nsize
```


- 131 -

```
      do 30 j=1,nsize
      if(h(i,j).eq.1) goto 40
      goto 30
 40   continue
      p(2)=h(i,j+1)
      p(3)=h(i-1,j+1)
      p(4)=h(i-1,j)
      p(5)=h(i-1,j-1)
      p(6)=h(i,j-1)
      p(7)=h(i+1,j-1)
      p(8)=h(i+1,j)
      p(9)=h(i+1,j+1)
      g(2)=p(2)
      g(3)=p(3)
      g(4)=p(4)
      g(5)=p(5)
      g(6)=p(6)
      g(7)=p(7)
      g(8)=p(8)
      g(9)=p(9)
c the first condition
      sum=p(2)+p(4)+p(6)+p(8)
      if(sum.le.3) goto 50
      goto 30
 50   continue
c the second condition
      init1=0
      do 1 k=2,9
      if(p(k).eq.1) then
      init1=init1+1
      endif
 1    continue
      if(init1.ge.2) goto 60
      goto 30
 60   continue
c the third condition
      init2=0
      do 2 k=2,9
      if(g(k).eq.1) then
      init2=init2+1
```

```
      endif
 2    continue
      if(init2.ge.1) goto 70
      goto 30
 70   continue
c the fourth condition
      do 3 m=2,5
      l=2*m-2
      if(p(l).eq.0) goto 80
      b(m)=0
      goto 3
 80   ll=2*m-1
      if(p(ll).eq.1.or.p(ll+1).eq.1) goto 90
      b(m)=0
      goto 3
 90   continue
      b(m)=1
 3    continue
      xp1=0
      do 4 n=2,5
      xp1=xp1+b(n)
 4    continue
      if(xp1.eq.1) goto 100
      goto 30
 100  continue
c the fifth condition
      if(g(4).eq.1.or.g(4).eq.0) goto 110
      goto 30
 110  continue
c the sixth condition
      if(g(6).eq.1.or.g(6).eq.0) goto 120
      goto 30
 120  continue
      h(i,j)=2
 30   continue
      do 200 i=1,nsize
      do 200 j=1,nsize
      if(h(i,j).eq.2) then
      h(i,j)=0
      endif
```

```
200   continue
7     continue
      do 131 i=1,nsize
      do 131 j=1,nsize
      if(h(i,j).eq.0) then
      h(i,j)=255
      else
      h(i,j)=0
      endif
131   continue
      do 140 i=1,nsize
      do 140 j=1,nsize
      img(i,j)=char(h(i,j))
140   continue
      write(2) ((img(i,j),j=1,nsize),i=1,nsize)
      close(2)
      stop
      end
```

```
C*****************************************************************
c   filename : algsub10.for                                       *
c                                                                 *
c Description : This program is implemented to perform           *
c               thinning algorithm using the "simplified          *
c               version of Hilditch". The algorithm               *
c               consists of four conditions and if they           *
c               are satisfied the point p1 is deleted             *
c                                                                 *
C*****************************************************************
$large
      integer*2 h(128,128),g(128,128),cc,kk
      character img(128,128)
      integer nsize,m,c,p(9),pro1,pro2,pro3,pro4,mm
      character*16 filnme,filename
      write(*,*) 'enter the input filename'
      read(*,'(a16)') filnme
      write(*,*) 'enter the output filename'
      read(*,'(a16)') filename
      write(*,*)' enter size of the image '
      read(*,*) nsize
      write(*,*)' 1'
      open(1,file=filnme,form='binary',status='old')
      open(2,file=filename,form='binary',status='new')
      write(*,*)' 2'
      do 1 i=1,nsize
      do 1 j=1,nsize
      read(1) img(i,j)
      h(i,j)=ichar(img(i,j))
      if(h(i,j).eq.0) then
      h(i,j)=1
      else
      h(i,j)=0
      endif
 1    continue
      close(1)
      write(*,*) ' enter cc '
      read(*,*) cc
      do 7 kk=1,cc
```

```
      do 2 i=1,nsize
      do 2 j=1,nsize
      if (h(i,j).eq.1) goto 15
      goto 2
15    p(1)=h(i,j)
      p(2)=h(i-1,j)
      p(3)=h(i-1,j+1)
      p(4)=h(i,j+1)
      p(5)=h(i+1,j+1)
      p(6)=h(i+1,j)
      p(7)=h(i+1,j-1)
      p(8)=h(i,j-1)
      p(9)=h(i-1,j-1)
      m=p(2)+p(3)+p(4)+p(5)+p(6)+p(7)+p(8)+p(9)
      if(m.le.6.and.m.ge.2) goto 11
      goto 2
11    init=0
      do 10 mm=2,8
      if(p(mm).eq.0.and.p(mm+1).eq.1) then
      init=init+1
      endif
10    continue
      if(p(9).eq.0.and.p(2).eq.1) then
      init=init+1
      endif
      if(init.eq.1) goto 12
      goto 2
12    pro1=p(2)*p(4)*p(6)
      pro2=p(2)*p(4)*p(8)
      if(pro1.eq.0.and.pro2.eq.0) goto 3
      goto 2
3     h(i,j)=0
2     continue
7     continue
      do 13 i=1,nsize
      do 13 j=1,nsize
      if(h(i,j).eq.0) then
      h(i,j)=255
      else
      h(i,j)=0
      endif
13    continue
```

```
      do 8 i=1,nsize
      do 8 j=1,nsize
      img(i,j)=char(h(i,j))
8     continue
      write(2)((img(i,j),j=1,nsize),i=1,nsize)
      close(2)
      stop
      end
```

```
c*****************************************************************************
c  filename :-algsub1.for                                                    *
c                                                                            *
c  Description : This program is implemented to perform                     *
c                thinning algorithm using the "simplified                    *
c                version of Hilditch" . In each pass the                     *
c                the point p1 is flagged and at the end of                   *
c                the pass all flagged points are deleted.                    *
c                                                                            *
c*****************************************************************************
$large
      integer*2 h(128,128),g(128,128),cc,kk
      character img(128,128)
      integer nsize,m,c,p(9),pro1,pro2,pro3,pro4,mm
      character*16 filnme,filename
      write(*,*) 'enter the input filename'
      read(*,'(a16)') filnme
      write(*,*) 'enter the output filename'
      read(*,'(a16)') filename
      write(*,*)' enter size of the image '
      read(*,*) nsize
      write(*,*)' 1'
      open(1,file=filnme,form='binary',status='old')
      open(2,file=filename,form='binary',status='new')
      write(*,*)' 2'
      do 1 i=1,nsize
      do 1 j=1,nsize
      read(1) img(i,j)
      h(i,j)=ichar(img(i,j))
      if(h(i,j).eq.0) then
      h(i,j)=1
      g(i,j)=1
      else
      h(i,j)=0
      g(i,j)=0
      endif
1     continue
      close(1)
      write(*,*) ' enter cc '
      read(*,*) cc
```

```
      do 7 kk=1,cc
      do 2 i=1,nsize
      do 2 j=1,nsize
      if (h(i,j).eq.1) goto 15
      goto 2
15    p(1)=h(i,j)
      p(2)=h(i-1,j)
      p(3)=h(i-1,j+1)
      p(4)=h(i,j+1)
      p(5)=h(i+1,j+1)
      p(6)=h(i+1,j)
      p(7)=h(i+1,j-1)
      p(8)=h(i,j-1)
      p(9)=h(i-1,j-1)
      m=p(2)+p(3)+p(4)+p(5)+p(6)+p(7)+p(8)+p(9)
      if(m.le.6.and.m.ge.2) goto 11
      goto 2
11    init=0
      do 10 mm=2,8
      if(p(mm).eq.0.and.p(mm+1).eq.1) then
      init=init+1
      endif
10    continue
      if(p(9).eq.0.and.p(2).eq.1) then
      init=init+1
      endif
      if(init.eq.1) goto 12
      goto 2
12    pro1=p(2)*p(4)*p(6)
      pro2=p(2)*p(4)*p(8)
      if(pro1.eq.0.and.pro2.eq.0) goto 3
      goto 2
3     g(i,j)=2
2     continue
      do 130 ii=1,nsize
      do 130 jj=1,nsize
      if(g(ii,jj).eq.2) then
      h(ii,jj)=0
      endif
130   continue
7     continue
      do 13 i=1,nsize
```

```
      do 13 j=1,nsize
      if(h(i,j).eq.0) then
      h(i,j)=255
      else
      h(i,j)=0
      endif
13    continue
      do 8 i=1,nsize
      do 8 j=1,nsize
      img(i,j)=char(h(i,j))
8     continue
      write(2)((img(i,j),j=1,nsize),i=1,nsize)
      close(2)
      stop
      end
```

```
C*********************************************************************
c  filename : d.for                                                   *
c     Description : This program is implemented to perform           *
c                   thinning algorithm using Lee's method            *
c                   based on some aspects which can be               *
c                   summarized as follows :                          *
c                     1. Distance transform                          *
c                     2. New selection rules using the first         *
c                        equation.                                   *
c                     3. Linking algorithm                           *
c                     4. Thinning algorithm                          *
C*********************************************************************
      integer h(128,128),g(128,128),d2(128,128),d3(128,128)
      integer t(8),tm(8),q(8),tt(8),m,k,or,m1,m2,dd
      character img(128,128)
      character*16 filnme,filename
      write(*,*) 'enter the input filename'
      read(*,'(a16)') filnme
      write(*,*) 'enter the output filename'
      read(*,'(a16)') filename
      open(1,file=filnme,form='binary',status='old')
      open(2,file=filename,form='binary',status='new')
      do 1 i=1,128
      do 1 j=1,128
      read(1) img(i,j)
      h(i,j)=ichar(img(i,j))
1     continue
      close(1)
      do 2 i=1,128
      do 2 j=1,128
      if(h(i,j).eq.0) then
      h(i,j)=1
      else
      h(i,j)=0
      endif
2     continue
c   Distance transform
      do 3 i=1,128
      do 3 j=1,128
```

```
      if(h(i,j).eq.1) then
      do 4 k=1,128
      if(h(i,j).ne.h(i-k,j).or.h(i,j).ne.h(i+k,j)) then
      max1=k
      goto 5
      endif
 4    continue
 5    continue
      do 41 m=1,128
      if(h(i,j).ne.h(i,j-m).or.h(i,j).ne.h(i,j+m)) then
      max2=m
      goto 42
      endif
 41   continue
 42   continue
      if(max1.lt.max2) then
      g(i,j)=max1
      else
      g(i,j)=max2
      endif
      endif
 3    continue
c the new selection rules
      do 43 i=1,128
      do 43 j=1,128
      if(g(i,j)+1.gt.g(i,j-1).and.g(i,j)+1.gt.g(i-1,j)) goto
      goto 44
 45   if(g(i,j)+1.gt.g(i,j+1).and.g(i,j)+1.gt.g(i+1,j)) goto
      goto 44
 46   h(i,j)=g(i,j)
      goto 43
 44   h(i,j)=0
 43   continue
      do 17 i=1,128
      do 17 j=1,128
      d2(i,j)=h(i,j)
 17   continue
      do 14 i=1,128
      do 14 j=1,128
      if(h(i,j).eq.0) then
      d2(i,j)=0
```

```
        else
        d2(i,j)=1
        endif
14      continue
        do 58 i=1,128
        do 58 j=1,128
        if(h(i,j).eq.1) then
        g(i,j)=h(i-1,j-1)+h(i-1,j)+h(i-1,j+1)+h(i,j+1)
     *  +h(i+1,j+1)+h(i+1,j)+h(i+1,j-1)+h(i,j-1)
        if(g(i,j).eq.0) then
        d2(i,j)=0
        endif
        endif
58      continue
c linking algorithm
        do 123 i=5,128
        do 21 j=2,127
        if(d2(i,j).ne.0) then
c SUBPROGRAM A :
        if((d2(i-1,j-1).ne.0.and.d2(i+1,j+1).ne.0).or.
     *  (d2(i,j-1).ne.0.and.d2(i,j+1).ne.0).or.
     *  (d2(i+1,j-1).ne.0.and.d2(i-1,j+1).ne.0).or.
     *  (d2(i-1,j).ne.0.and.d2(i+1,j).ne.0).or.
     *  (d2(i-1,j-1).ne.0.and.d2(i-1,j+1).ne.0).or.
     *  (d2(i-1,j+1).ne.0.and.d2(i+1,j+1).ne.0).or.
     *  (d2(i+1,j-1).ne.0.and.d2(i+1,j+1).ne.0).or.
     *  (d2(i-1,j-1).ne.0.and.d2(i+1,j-1).ne.0)) then
        lt=1
        else
        lt=0
        endif
        if(lt.eq.0) then
c SUBPROGRAM B :
c n region
         if(d2(i-1,j).eq.0) then
          if(d2(i-2,j).ne.0.or.d2(i-3,j).ne.0) then
            d3(i-1,j)=g(i-1,j)
             if(d2(i-3,j).ne.0) goto 21
          endif
         endif
c ne region
```

```
        if(d2(i-1,j).eq.0.and.d2(i-1,j+1).eq.0.and.
     * d2(i,j+1).eq.0) then
        if(d2(i-3,j+1).ne.0.or.d2(i-3,j+2).ne.0.or.
     * d2(i-3,j+3).ne.0.or.d2(i-2,j+3).ne.0.or.
     *  d2(i-1,j+3).ne.0) then
          d3(i-1,j+1)=g(i-1,j+1)
          goto 21
        endif
        endif
c e region
        if(d2(i,j+1).eq.0) then
          if(d2(i,j+2).ne.0.or.d2(i,j+3).ne.0.) then
           d3(i,j+1)=g(i,j+1)
        if(d2(i,j+3).ne.0) goto 21
          endif
        endif
c se region
        if(d2(i,j+1).eq.0.and.d2(i+1,j+1).eq.0.and.d2(i+1,j).
     *  eq.0.and.d2(i+2,j).eq.0.and.d2(i+3,j).eq.0) then
       if(d2(i+1,j+3).ne.0.or.d2(i+2,j+3).ne.0.or.d2(i+3,j+3).
     *  ne.0.or.d2(i+3,j+2).ne.0.or.d2(i+3,j+1).ne.0) then
        d3(i+1,j+1)=g(i+1,j+1)
        goto 21
        endif
        endif
c s region
        if(d2(i+1,j+1).eq.0.and.d2(i+1,j).eq.0) then
          if(d2(i+3,j).ne.0) then
           d3(i+1,j)=g(i+1,j )
          endif
        endif
        endif
        endif
 21      continue
 123     continue
c        thinning algorithm
        do 7 i=1,128
        do 7 j=1,128
        g(i,j)=d2(i,j)+d3(i,j)
        if(g(i,j).eq.0) then
        h(i,j)=0
```

```
        else
        h(i,j)=1
        endif
7       continue
        write(*,*)'enter dd '
        read(*,*) dd
        do 99 kk=1,dd
        do 20 i=1,128
        do 20 j=1,128
        if(h(i,j).eq.1) goto 100
        goto 20
100     t(1)=h(i-1,j)
        t(2)=h(i-1,j+1)
        t(3)=h(i,j+1)
        t(4)=h(i+1,j+1)
        t(5)=h(i+1,j)
        t(6)=h(i+1,j-1)
        t(7)=h(i,j-1)
        t(8)=h(i-1,j-1)
c tm
        tm(1)=t(1)
        tm(2)=0
        tm(3)=t(3)
        tm(4)=0
        tm(5)=t(5)
        tm(6)=0
        tm(7)=t(7)
        tm(8)=0
c Shift tm
        q(1)=0
        q(2)=tm(1)
        q(3)=tm(2)
        q(4)=tm(3)
        q(5)=tm(4)
        q(6)=tm(5)
        q(7)=tm(6)
        q(8)=tm(7)
c OR_ing t and q
        do 200 k=1,8
        or=t(k)+q(k)
        if(or.eq.0) then
        tt(k)=0.
```

```
        else
        tt(k)=1
        endif
200     continue
        n1tt=tt(1)+tt(2)+tt(3)+tt(4)+tt(5)+tt(6)+tt(7)+tt(8)
        n10tt=0
        do 300 m=2,8
        if(tt(m-1).eq.1.and.tt(m).eq.0) then
        n10tt=n10tt+1
        endif
300     continue
        if(tt(8).eq.1.and.tt(1).eq.0) then
        n10tt=n10tt+1
        endif
        if(n10tt.gt.1) goto 1000
        goto 1001
1000    h(i,j)=1
1001    if(n10tt.eq.1) goto 1002
        goto 20
1002    if(n1tt.eq.1) then
        h(i,j)=1
        else
        h(i,j)=0
        endif
20      continue
99      continue
        do 40 i=1,128
        do 40 j=1,128
        if(h(i,j).eq.0) then
        h(i,j)=255
        else
        h(i,j)=0
        endif
40      continue
        do 50 i=1,128
        do 50 j=1,128
        img(i,j)=char(h(i,j))
50      continue
        write(2) ((img(i,j),j=1,128),i=1,128)
        close(2)
        stop
        end
```

```
c*****************************************************************
c.   filename : d1.for                                            *
c                                                                 *
c    Description : This program is implemented to perform        *
c                  thinning algorithm using Lee's method        *
c                  based on some concepts which can be          *
c                  summerized as follows.                       *
c                      1. Distance transform                    *
c                      2. New selection rules using the         *
c                        second equation .                      *
c                      3. linking algorithm                     *
c                      4. thinning algorithm                    *
c                                                               *
c*****************************************************************
      integer h(128,128),g(128,128),d2(128,128),d3(128,128)
      integer t(8),tm(8),q(8),tt(8),m,k,or,m1,m2,mm
      character img(128,128)
      character*16 filnme,filename
      write(*,*) 'enter the input filename'
      read(*,'(a16)') filnme
      write(*,*) 'enter the output filename'
      read(*,'(a16)') filename
      open(1,file=filnme,form='binary',status='old')
      open(2,file=filename,form='binary',status='new')
      do 1 i=1,128
      do 1 j=1,128
      read(1) img(i,j)
      h(i,j)=ichar(img(i,j))
1     continue
      close(1)
      do 2 i=1,128
      do 2 j=1,128
      if(h(i,j).eq.0) then
      h(i,j)=1
      else
      h(i,j)=0
      endif
2     continue
c  Distance transform
      do 3 i=1,128
```

```
      do 3 j=1,128
      if(h(i,j).eq.1) then
      do 4 k=1,128
      if(h(i,j).ne.h(i-k,j).or.h(i,j).ne.h(i+k,j)) then
      max1=k
      goto 5
      endif
4     continue
5     continue
      do 41 m=1,128
      if(h(i,j).ne.h(i,j-m).or.h(i,j).ne.h(i,j+m)) then
      max2=m
      goto 42
      endif
41    continue
42    continue
      if(max1.lt.max2) then
      g(i,j)=max1
      else
      g(i,j)=max2
      endif
      endif
3     continue
c  New selection rules
      do 43 i=1,128
      do 43 j=1,128
      if((g(i,j)-g(i,j-2))+(g(i,j)-g(i-2,j))+(g(i,j)
     +  -g(i+2,j))+(g(i,j)-g(i,j+2)).gt.2) goto 46
      goto 44
46    h(i,j)=g(i,j)
      goto 43
44    h(i,j)=0
43    continue
      do 14 i=1,128
      do 14 j=1,128
      if(h(i,j).eq.0) then
      d2(i,j)=0
      else
      d2(i,j)=1
      endif
14    continue
```

```
      do 58 i=1,128
      do 58 j=1,128
      if(h(i,j).eq.1) then
      g(i,j)=h(i-1,j-1)+h(i-1,j)+h(i-1,j+1)+h(i,j+1)
     *  +h(i+1,j+1)+h(i+1,j)+h(i+1,j-1)+h(i,j-1)
      if(g(i,j).eq.0) then
      d2(i,j)=0
      endif
      endif
58    continue
c . Linking algorithm
      do 123 i=5,128
      do 21  j=2,127
      if(d2(i,j).ne.0) then
c SUBPROGRAM A :
      if((d2(i-1,j-1).ne.0.and.d2(i+1,j+1).ne.0).or.
     *  (d2(i,j-1).ne.0.and.d2(i,j+1).ne.0).or.
     *  (d2(i+1,j-1).ne.0.and.d2(i-1,j+1).ne.0).or.
     *  (d2(i-1,j).ne.0.and.d2(i+1,j).ne.0).or.
     *  (d2(i-1,j-1).ne.0.and.d2(i-1,j+1).ne.0).or.
     *  (d2(i-1,j+1).ne.0.and.d2(i+1,j+1).ne.0).or.
     *  (d2(i+1,j-1).ne.0.and.d2(i+1,j+1).ne.0).or.
     *  (d2(i-1,j-1).ne.0.and.d2(i+1,j-1).ne.0)) then
      lt=1
      else
      lt=0
      endif
      if(lt.eq.0) then
c SUBPROGRAM B :
c n region
       if(d2(i-1,j).eq.0) then
        if(d2(i-2,j).ne.0.or.d2(i-3,j).ne.0) then
          d3(i-1,j)=g(i-1,j)
           if(d2(i-3,j).ne.0) goto 21
        endif
       endif
c ne region
      if(d2(i-1,j).eq.0.and.d2(i-1,j+1).eq.0.and.
     *  d2(i,j+1).eq.0) then
      if(d2(i-3,j+1).ne.0.or.d2(i-3,j+2).ne.0.or.
     *  d2(i-3,j+3).ne.0.or.d2(i-2,j+3).ne.0.or.
```

```
     *   d2(i-1,j+3).ne.0) then
         d3(i-1,j+1)=g(i-1,j+1)
         goto 21
        endif
        endif
c e region
        if(d2(i,j+1).eq.0) then
          if(d2(i,j+2).ne.0.or.d2(i,j+3).ne.0.) then
           d3(i,j+1)=g(i,j+1)
        if(d2(i,j+3).ne.0) goto 21
          endif
        endif
c se region
        if(d2(i,j+1).eq.0.and.d2(i+1,j+1).eq.0.and.d2(i+1,j).
     *  eq.0.and.d2(i+2,j).eq.0.and.d2(i+3,j).eq.0) then
        if(d2(i+1,j+3).ne.0.or.d2(i+2,j+3).ne.0.or.d2(i+3,j+3).
     *  ne.0.or.d2(i+3,j+2).ne.0.or.d2(i+3,j+1).ne.0) then
        d3(i+1,j+1)=g(i+1,j+1)
        goto 21
        endif
        endif
c s region
        if(d2(i+1,j+1).eq.0.and.d2(i+1,j).eq.0) then
          if(d2(i+3,j).ne.0) then
           d3(i+1,j)=g(i+1,j )
          endif
        endif
        endif
        endif
 21      continue
 123     continue
c        thinning algorithm
        do 7 i=1,128
        do 7 j=1,128
c        g(i,j)=d3(i,j)+d2(i,j)
        g(i,j)=d2(i,j)+d3(i,j)
        if(g(i;j).eq.0) then
        h(i,j)=0
        else
        h(i,j)=1
        endif
```

```
7       continue
        do 99 kk=1,10
        do 20 i=1,128
        do 20 j=1,128
        if(h(i,j).eq.1) goto 100
        goto 20
100     t(1)=h(i-1,j)
        t(2)=h(i-1,j+1)
        t(3)=h(i,j+1)
        t(4)=h(i+1,j+1)
        t(5)=h(i+1,j)
        t(6)=h(i+1,j-1)
        t(7)=h(i,j-1)
        t(8)=h(i-1,j-1)
c tm
        tm(1)=t(1)
        tm(2)=0
        tm(3)=t(3)
        tm(4)=0
        tm(5)=t(5)
        tm(6)=0
        tm(7)=t(7)
        tm(8)=0
c Shift tm
        q(1)=0
        q(2)=tm(1)
        q(3)=tm(2)
        q(4)=tm(3)
        q(5)=tm(4)
        q(6)=tm(5)
        q(7)=tm(6)
        q(8)=tm(7)
c OR_ing t and q
        do 200 k=1,8
        or=t(k)+q(k)
        if(or.eq.0) then
        tt(k)=0
        else
        tt(k)=1
        endif
200     continue
        n1tt=tt(1)+tt(2)+tt(3)+tt(4)+tt(5)+tt(6)+tt(7)+tt(8)
```

```
      n10tt=0
      do 300 m=2,8
      if(tt(m-1).eq.1.and.tt(m).eq.0) then
      n10tt=n10tt+1
      endif
300   continue
      if(tt(8).eq.1.and.tt(1).eq.0) then
      n10tt=n10tt+1
      endif
      if(n10tt.gt.1) goto 1000
      goto 1001
1000  h(i,j)=1
1001  if(n10tt.eq.1) goto 1002
      goto 20
1002  if(n1tt.eq.1) then
      h(i,j)=1
      else
      h(i,j)=0
      endif
20    continue
      do 40 i=1,128
      do 40 j=1,128
      if(h(i,j).eq.0) then
      h(i,j)=255
      else
      h(i,j)=0
      endif
40    continue
      do 50 i=1,128
      do 50 j=1,128
      img(i,j)=char(h(i,j))
50    continue
      write(2) ((img(i,j),j=1,128),i=1,128)
      close(2)
      stop
      end
```

```
c*****************************************************************
c  filename : álgsub4.for                                        *
c                                                                *
c   Description : This Program is implemented to perform        *
c                 thinning algorithm using local operations      *
c                 to detect edge-points , end-points and         *
c                 break-points. The algorithm is divided         *
c                 into four sub-iterations                       *
c*****************************************************************
$large
      integer*2 h(128,128),g(128,128),cc,kk,pro5,pro6,pro7
      character img(128,128)
      integer nsize,m,c,p(9),pro1,pro2,pro3,pro4,mm,pro8
      character*16 filnme,filename
      write(*,*) 'enter the input filename'
      read(*,'(a16)') filnme
      write(*,*) 'enter the output filename'
      read(*,'(a16)') filename
      write(*,*)' enter size of the image '
      read(*,*) nsize
      write(*,*)' 1'
      open(1,file=filnme,form='binary',status='old')
      open(2,file=filename,form='binary',status='new')
      write(*,*)' 2'
      do 1 i=1,nsize
      do 1 j=1,nsize
      read(1) img(i,j)
      h(i,j)=ichar(img(i,j))
      if(h(i,j).eq.0) then
      h(i,j)=1
      g(i,j)=1
      else
      h(i,j)=0
      g(i,j)=0
      endif
1     continue
      close(1)
      write(*,*) ' enter cc '
      read(*,*) cc
      do 7 kk=1,cc
```

```
c*********************************************
c the first subiteration
c*********************************************
      do 2 i=1,nsize
      do 2 j=1,nsize
      if (h(i,j).eq.1) goto 15
      goto 2
 15   p(1)=h(i,j)
      p(2)=h(i-1,j)
      p(3)=h(i-1,j+1)
      p(4)=h(i,j+1)
      p(5)=h(i+1,j+1)
      p(6)=h(i+1,j)
      p(7)=h(i+1,j-1)
      p(8)=h(i,j-1)
      p(9)=h(i-1,j-1)
      m=p(2)+p(3)+p(4)+p(5)+p(6)+p(7)+p(8)+p(9)
      if(m.le.6.and.m.ge.2) goto 11
      goto 2
 11   init=0
      do 10 mm=2,8
      if(p(mm).eq.0.and.p(mm+1).eq.1) then
      init=init+1
      endif
 10   continue
      if(p(9).eq.0.and.p(2).eq.1) then
      init=init+1
      endif
      if(init.eq.1) goto 12
      goto 2
 12   pro1=p(2)*p(4)*p(6)
      pro2=p(4)*p(6)*p(8)
      if(pro1.eq.0.and.pro2.eq.0) goto 3
      goto 2
 3    g(i,j)=2
 2    continue
      do 6 i=1,nsize
      do 6 j=1,nsize
      if(g(i,j).eq.2) then
      h(i,j)=0
      endif
 6    continue
```

```
c*********************************
c The second subiteration
c*********************************
      do 21 i=1,nsize
      do 21 j=1,nsize
      if (h(i,j).eq.1) goto 151
      goto 21
151   p(1)=h(i,j)
      p(2)=h(i-1,j)
      p(3)=h(i-1,j+1)
      p(4)=h(i,j+1)
      p(5)=h(i+1,j+1)
      p(6)=h(i+1,j)
      p(7)=h(i+1,j-1)
      p(8)=h(i,j-1)
      p(9)=h(i-1,j-1)
      m=p(2)+p(3)+p(4)+p(5)+p(6)+p(7)+p(8)+p(9)
      if(m.le.6.and.m.ge.2) goto 111
      goto 21
111   init=0
      do 101 mm=2,8
      if(p(mm).eq.0.and.p(mm+1).eq.1) then
      init=init+1
      endif
101   continue
      if(p(9).eq.0.and.p(2).eq.1) then
      init=init+1
      endif
      if(init.eq.1) goto 121
      goto 21
121   pro3=p(2)*p(6)*p(8)
      pro4=p(4)*p(6)*p(8)
      if(pro3.eq.0.and.pro4.eq.0) goto 31
      goto 21
31    g(i,j)=2
21    continue
      do 61 i=1,nsize
      do 61 j=1,nsize
      if(g(i,j).eq.2) then
      h(i,j)=0
      endif
61    continue
```

```
c*********************************
c the third subiteration
c*********************************
      do 50 k=1,nsize
      do 50 l=1,nsize
      if (h(k,l).eq.1) goto 60
      goto 50
 60   p(1)=h(k,l)
      p(2)=h(k-1,l)
      p(3)=h(k-1,l+1)
      p(4)=h(k,l+1)
      p(5)=h(k+1,l+1)
      p(6)=h(k+1,l)
      p(7)=h(k+1,l-1)
      p(8)=h(k,l-1)
      p(9)=h(k-1,l-1)
      m=p(2)+p(3)+p(4)+p(5)+p(6)+p(7)+p(8)+p(9)
      if(m.le.6.and.m.ge.2) goto 70
      goto 50
 70   init=0
      do 80 mm=2,8
      if(p(mm).eq.0.and.p(mm+1).eq.1) then
      init=init+1
      endif
 80   continue
      if(p(9).eq.0.and.p(2).eq.1) then
      init=init+1
      endif
      if(init.eq.1) goto 110
      goto 50
 110  pro5=p(2)*p(4)*p(8)
      pro6=p(2)*p(6)*p(8)
      if(pro5.eq.0.and.pro6.eq.0) goto 120
      goto 50
 120  g(k,l)=2
 50   continue
      do 130 ii=1,nsize
      do 130 jj=1,nsize
      if(g(ii,jj).eq.2) then
      h(ii,jj)=0
      endif
 130  continue
```

```
c*o*o*o*o*o*o*o*o*o*o*o*o*o*o*o*o*o*o*o*o*o*o*o*o*o*o*o*o*o*o*o*o*o*
c the fourth subiteration
c*o*o*o*o*o*o*o*o*o*o*o*o*o*o*o*o*o*o*o*o*o*o*o*o*o*o*o*o*o*o*o*o*o*
      do 501 k=1,nsize
      do 501 l=1,nsize
      if (h(k,l).eq.1) goto 601
      goto 501
601   p(1)=h(k,l)
      p(2)=h(k-1,l)
      p(3)=h(k-1,l+1)
      p(4)=h(k,l+1)
      p(5)=h(k+1,l+1)
      p(6)=h(k+1,l)
      p(7)=h(k+1,l-1)
      p(8)=h(k,l-1)
      p(9)=h(k-1,l-1)
      m=p(2)+p(3)+p(4)+p(5)+p(6)+p(7)+p(8)+p(9)
      if(m.le.6.and.m.ge.2) goto 701
      goto 501
701   init=0
      do 801 mm=2,8
      if(p(mm).eq.0.and.p(mm+1).eq.1) then
      init=init+1
      endif
801   continue
      if(p(9).eq.0.and.p(2).eq.1) then
      init=init+1
      endif
      if(init.eq.1) goto 1101
      goto 501
1101  pro7=p(2)*p(4)*p(6)
      pro8=p(2)*p(4)*p(8)
      if(pro7.eq.0.and.pro8.eq.0) goto 1201
      goto 501
1201  g(k,l)=2
501   continue
      do 1301 ii=1,nsize
      do 1301 jj=1,nsize
      if(g(ii,jj).eq.2) then
      h(ii,jj)=0
      endif
1301  continue
```

```
7       continue
        do 13 i=1,nsize
        do 13 j=1,nsize
        if(h(i,j).eq.0) then
        h(i,j)=255
        else
        h(i,j)=0
        endif
13      continue
        do 8 i=1,nsize
        do 8 j=1,nsize
        img(i,j)=char(h(i,j))
8       continue
        write(2)((img(i,j),j=1,nsize),i=1,nsize)
        close(2)
        stop
        end
```

```
c*****************************************************************
c  filename : temp2.for                                          *
c                                                                *
cDescription : In temp1.for using eight template and only       *
c              one image we obtain a skeleton but with many     *
c              branches in different parts of the character.    *
c              Temp2.for is implemented to remove this          *
c              disadvantage in which two images are used the    *
c              current image and the working image . Note       *
c              that the eight templates A1,B1,A2,B2,A3,B3,A4    *
c              and B4 are also used .                           *
c*****************************************************************
c
      integer h(128,128),g(128,128),p(9),nsize,k1,k2,kk
      character img(128,128)
      character*16 filnme,filename
      write(*,*)' enter the input filename '
      read(*,'(a16)') filnme
      write(*,*) 'enter the output filename'
      read(*,'(a16)') filename
      write(*,*)' enter the size of the image ---->'
      read(*,*) nsize
      write(*,*)' enter the scaling factors over X and Y '
      read(*,*) scalx,scaly
      open(1,file=filnme,form='binary',status='old')
      open(2,file=filename,form='binary',status='new')
      write(*,*)' 1'
      do 10 i=1,nsize
      do 10 j=1,nsize
      read(1) img(i,j)
      h(i,j)=ichar(img(i,j))
      if(h(i,j).eq.0)then
      h(i,j)=1
      else
      h(i,j)=0
      endif
10    continue
      write(*,*) ' 2'
      do 99 i=1,nsize
      do 99 j=1,nsize
```

```
          g(i,j)=h(i,j)
 99       continue
c template A1
          write(*,*) 'enter kk '
          read(*,*) kk
          do 101 k=1,kk
          do 1 i=1,nsize
          do 1 j=1,nsize
          if(h(i,j).eq.1) goto 11
          goto 1
 11       p(2)=h(i,j+1)
          p(3)=h(i-1,j+1)
          p(4)=h(i-1,j)
          p(5)=h(i-1,j-1)
          p(6)=h(i,j-1)
          p(7)=h(i+1,j-1)
          p(8)=h(i+1,j)
          p(9)=h(i+1,j+1)
          k1=p(4)+p(5)+p(6)
          k2=p(2)+p(8)
          if(k1.eq.0.and.k2.eq.2) then
          g(i,j)=0
          endif
 1        continue
          do 20 i=1,nsize
          do 20 j=1,nsize
          h(i,j)=g(i,j)
 20       continue
c template b1
          do 2 i=1,nsize
          do 2 j=1,nsize
          if(h(i,j).eq.1) goto 21
          goto 2
 21       p(2)=h(i,j+1)
          p(3)=h(i-1,j+1)
          p(4)=h(i-1,j)
          p(5)=h(i-1,j-1)
          p(6)=h(i,j-1)
          p(7)=h(i+1,j-1)
          p(8)=h(i+1,j)
          p(9)=h(i+1,j+1)
          k1=p(3)+p(4)+p(5)
```

```
      k2=p(7)+p(8)
      if(k1.eq.0.and.k2.eq.2) then
      g(i,j)=0
      endif
 2    continue
      do 30 i=1,nsize
      do 30 j=1,nsize
      h(i,j)=g(i,j)
 30   continue
c templates a2
      do 3 i=1,nsize
      do 3 j=1,nsize
      if(h(i,j).eq.1) goto 31
      goto 3
 31   p(2)=h(i,j+1)
      p(3)=h(i-1,j+1)
      p(4)=h(i-1,j)
      p(5)=h(i-1,j-1)
      p(6)=h(i,j-1)
      p(7)=h(i+1,j-1)
      p(8)=h(i+1,j)
      p(9)=h(i+1,j+1)
      k1=p(2)+p(3)+p(4)
      k2=p(6)+p(8)
      if(k1.eq.0.and.k2.eq.2) then
      g(i,j)=0
      endif
 3    continue
      do 40 i=1,nsize
      do 40 j=1,nsize
      h(i,j)=g(i,j)
 40   continue
c  template b2
      do 4 i=1,nsize
      do 4 j=1,nsize
      if(h(i,j).eq.1) goto 41
      goto 4
 41   p(2)=h(i,j+1)
      p(3)=h(i-1,j+1)
      p(4)=h(i-1,j)
      p(5)=h(i-1,j-1)
      p(6)=h(i,j-1)
```

```
      p(7)=h(i+1,j-1)
      p(8)=h(i+1,j)
      p(9)=h(i+1,j+1)
      k1=p(2)+p(3)+p(9)
      k2=p(5)+p(6)
      if(k1.eq.0.and.k2.eq.2) then
      g(i,j)=0
      endif
 4    continue
      do 50 i=1,nsize
      do 50 j=1,nsize
      h(i,j)=g(i,j)
 50   continue
c templates A3
      do 5 i=1,nsize
      do 5 j=1,nsize
      if(h(i,j).eq.1) goto 51
      goto 5
 51   p(2)=h(i,j+1)
      p(3)=h(i-1,j+1)
      p(4)=h(i-1,j)
      p(5)=h(i-1,j-1)
      p(6)=h(i,j-1)
      p(7)=h(i+1,j-1)
      p(8)=h(i+1,j)
      p(9)=h(i+1,j+1)
      k1=p(2)+p(8)+p(9)
      k2=p(4)+p(6)
      if(k1.eq.0.and.k2.eq.2) then
      g(i,j)=0
      endif
 5    continue
      do 60 i=1,nsize
      do 60 j=1,nsize
      h(i,j)=g(i,j)
 60   continue
c template b3
      do 6 i=1,nsize
      do 6 j=1,nsize
      if(h(i,j).eq.1) goto 61
      goto 6
 61   p(2)=h(i,j+1)
```

```
      p(3)=h(i-1,j+1)
      p(4)=h(i-1,j)
      p(5)=h(i-1,j-1)
      p(6)=h(i,j-1)
      p(7)=h(i+1,j-1)
      p(8)=h(i+1,j)
      p(9)=h(i+1,j+1)
      k1=p(7)+p(8)+p(9)
      k2=p(3)+p(4)
      if(k1.eq.0.and.k2.eq.2) then
      g(i,j)=0
      endif
 6    continue
      do 70 i=1,nsize
      do 70 j=1,nsize
      h(i,j)=g(i,j)
 70   continue
c templates A4
      do 7 i=1,nsize
      do 7 j=1,nsize
      if(h(i,j).eq.1) goto 71
      goto 7
 71   p(2)=h(i,j+1)
      p(3)=h(i-1,j+1)
      p(4)=h(i-1,j)
      p(5)=h(i-1,j-1)
      p(6)=h(i,j-1)
      p(7)=h(i+1,j-1)
      p(8)=h(i+1,j)
      p(9)=h(i+1,j+1)
      k1=p(6)+p(7)+p(8)
      k2=p(2)+p(4)
      if(k1.eq.0.and.k2.eq.2) then
      g(i,j)=0
      endif
 7    continue
      do 80 i=1,nsize
      do 80 j=1,nsize
      h(i,j)=g(i,j)
 80   continue
c  template b4
      do 8 i=1,nsize
```

```
      do 8 j=1,nsize
      if(h(i,j).eq.1) goto 81
      goto 8
81    p(2)=h(i,j+1)
      p(3)=h(i-1,j+1)
      p(4)=h(i-1,j)
      p(5)=h(i-1,j-1)
      p(6)=h(i,j-1)
      p(7)=h(i+1,j-1)
      p(8)=h(i+1,j)
      p(9)=h(i+1,j+1)
      k1=p(5)+p(6)+p(7)
      k2=p(2)+p(9)
      if(k1.eq.0.and.k2.eq.2) then
      g(i,j)=0
      endif
8     continue
      do 90 i=1,nsize
      do 90 j=1,nsize
      h(i,j)=g(i,j)
90    continue
101   continue
      do 100 i=1,nsize
      do 100 j=1,nsize
      if(g(i,j).eq.1) then
      g(i,j)=0
      else
      g(i,j)=255
      endif
100   continue
      do 110 i=1,nsize
      do 110 j=1,nsize
      img(i,j)=char(g(i,j))
110   continue
      write(2) ((img(i,j),j=1,nsize),i=1,nsize)
      close(2)
      stop
      end
```

```
c*****************************************************************
c    filename : tempwind.for                                     *
c                                                                *
c Description : In temp2.for (using the eight templates         *
c               and the two images ) we get an excellent        *
c               result in character without dots . But for      *
c               those with dots we still have branches on       *
c               the dots . Tempwind.for is implemented to       *
c               remove this problem using the following         *
c               procedure :                                     *
c                 1. Separate the dots from the characters      *
c                 2. Process the characters without dots        *
c                 3. Process the dots alone                     *
c                 4. Add the dots to the character              *
c*****************************************************************
      integer h(128,128),d(128,128),g(128,128),p(9),k1,k2,kk,
      integer hag(3),mag(3)
      character img(128,128)
      character*16 filnme,filename
      write(*,*)' enter the input filename '
      read(*,'(a16)') filnme
      write(*,*) 'enter the output filename'
      read(*,'(a16)') filename
      write(*,*)' enter the size of the image ---->'
      read(*,*) nsize
      open(1,file=filnme,form='binary',status='old')
      open(2,file=filename,form='binary',status='new')
      write(*,*)' 1'
      do 200 i=1,128
      do 200 j=1,128
      d(i,j)=0
200   continue
      imp=0
      do 201 i=1,128
      do 201 j=1,128
      read(1) img(i,j)
      h(i,j)=ichar(img(i,j))
      if(h(i,j).eq.0) then
      h(i,j)=1
      else
```

```
        h(i,j)=0
        endif
201     continue
        close(1)
c Separate the dots from the characters
        write(*,*)'enter the length'
        read(*,*) k1
        k2=int(k1/2.0)
        do 203 i=k2+1,128-k2
        do 203 j=k2+1,128-k2
        if(h(i,j).eq.1) then
        sum=0
        do 204 k=1,k2
        a1=h(i-k2,j)
        a2=h(i-k2,j-k)
        a3=h(i-k2,j+k)
        a4=h(i+k2,j)
        a5=h(i+k2,j-k)
        a6=h(i+k2,j+k)
        a7=h(i,j-k2)
        a8=h(i-k,j-k2)
        a9=h(i+k,j-k2)
        a10=h(i,j+k2)
        a11=h(i-k,j+k2)
        a12=h(i+k,j+k2)
        a=a1+a2+a3+a4+a5+a6+a7+a8+a9+a10+a11+a12
        sum=sum+a
204       continue
        if(sum.eq.0) then
        do 205 ii=i,i+k1
        do 205 jj=j,j+k1
        if(h(ii-k2-1,jj-k2-1).eq.1) then
        d(ii-k2-1,jj-k2-1)=1
        h(ii-k2-1,jj-k2-1)=0
        endif
205     continue
        imp=imp+1
        mag(imp)=i
        hag(imp)=j
        endif
        endif
```

```
203     continue
        do 206 i=1,nsize
        do 206 j=1,nsize
        g(i,j)=h(i,j)
206     continue
c Process the characters without dots
c template A1
        write(*,*) 'enter kk '
        read(*,*) kk
        do 101 k=1,kk
        do 1 i=1,nsize
        do 1 j=1,nsize
        if(h(i,j).eq.1) goto 11
        goto 1
11      p(2)=h(i,j+1)
        p(3)=h(i-1,j+1)
        p(4)=h(i-1,j)
        p(5)=h(i-1,j-1)
        p(6)=h(i,j-1)
        p(7)=h(i+1,j-1)
        p(8)=h(i+1,j)
        p(9)=h(i+1,j+1)
        k1=p(4)+p(5)+p(6)
        k2=p(2)+p(8)
        if(k1.eq.0.and.k2.eq.2) then
        g(i,j)=0
        endif
1       continue
        do 20 i=1,nsize
        do 20 j=1,nsize
        h(i,j)=g(i,j)
20      continue
c template b1
        do 2 i=1,nsize
        do 2 j=1,nsize
        if(h(i,j).eq.1) goto 21
        goto 2
21      p(2)=h(i,j+1)
        p(3)=h(i-1,j+1)
        p(4)=h(i-1,j)
        p(5)=h(i-1,j-1)
```

```
      p(6)=h(i,j-1)
      p(7)=h(i+1,j-1)
      p(8)=h(i+1,j)
      p(9)=h(i+1,j+1)
      k1=p(3)+p(4)+p(5)
      k2=p(7)+p(8)
      if(k1.eq.0.and.k2.eq.2) then
      g(i,j)=0
      endif
2     continue
      do 30 i=1,nsize
      do 30 j=1,nsize
      h(i,j)=g(i,j)
30    continue
c templates a2
      do 3 i=1,nsize
      do 3 j=1,nsize
      if(h(i,j).eq.1) goto 31
      goto 3
31    p(2)=h(i,j+1)
      p(3)=h(i-1,j+1)
      p(4)=h(i-1,j)
      p(5)=h(i-1,j-1)
      p(6)=h(i,j-1)
      p(7)=h(i+1,j-1)
      p(8)=h(i+1,j)
      p(9)=h(i+1,j+1)
      k1=p(2)+p(3)+p(4)
      k2=p(6)+p(8)
      if(k1.eq.0.and.k2.eq.2) then
      g(i,j)=0
      endif
3     continue
      do 40 i=1,nsize
      do 40 j=1,nsize
      h(i,j)=g(i,j)
40    continue
c  template b2
      do 4 i=1,nsize
      do 4 j=1,nsize
      if(h(i,j).eq.1) goto 41
      goto 4
```

```
41      p(2)=h(i,j+1)
        p(3)=h(i-1,j+1)
        p(4)=h(i-1,j)
        p(5)=h(i-1,j-1)
        p(6)=h(i,j-1)
        p(7)=h(i+1,j-1)
        p(8)=h(i+1,j)
        p(9)=h(i+1,j+1)
        k1=p(2)+p(3)+p(9)
        k2=p(5)+p(6)
        if(k1.eq.0.and.k2.eq.2) then
        g(i,j)=0
        endif
 4      continue
        do 50 i=1,nsize
        do 50 j=1,nsize
        h(i,j)=g(i,j)
 50     continue
c templates A3
        do 5 i=1,nsize
        do 5 j=1,nsize
        if(h(i,j).eq.1) goto 51
        goto 5
 51     p(2)=h(i,j+1)
        p(3)=h(i-1,j+1)
        p(4)=h(i-1,j)
        p(5)=h(i-1,j-1)
        p(6)=h(i,j-1)
        p(7)=h(i+1,j-1)
        p(8)=h(i+1,j)
        p(9)=h(i+1,j+1)
        k1=p(2)+p(8)+p(9)
        k2=p(4)+p(6)
        if(k1.eq.0.and.k2.eq.2) then
        g(i,j)=0
        endif
 5      continue
        do 60 i=1,nsize
        do 60 j=1,nsize
        h(i,j)=g(i,j)
 60     continue
c template b3
```

```
      do 6 i=1,nsize
      do 6 j=1,nsize
      if(h(i,j).eq.1) goto 61
      goto 6
61    p(2)=h(i,j+1)
      p(3)=h(i-1,j+1)
      p(4)=h(i-1,j)
      p(5)=h(i-1,j-1)
      p(6)=h(i,j-1)
      p(7)=h(i+1,j-1)
      p(8)=h(i+1,j)
      p(9)=h(i+1,j+1)
      k1=p(7)+p(8)+p(9)
      k2=p(3)+p(4)
      if(k1.eq.0.and.k2.eq.2) then
      g(i,j)=0
      endif
6     continue
      do 70 i=1,nsize
      do 70 j=1,nsize
      h(i,j)=g(i,j)
70    continue
c templates A4
      do 7 i=1,nsize
      do 7 j=1,nsize
      if(h(i,j).eq.1) goto 71
      goto 7
71    p(2)=h(i,j+1)
      p(3)=h(i-1,j+1)
      p(4)=h(i-1,j)
      p(5)=h(i-1,j-1)
      p(6)=h(i,j-1)
      p(7)=h(i+1,j-1)
      p(8)=h(i+1,j)
      p(9)=h(i+1,j+1)
      k1=p(6)+p(7)+p(8)
      k2=p(2)+p(4)
      if(k1.eq.0.and.k2.eq.2) then
      g(i,j)=0
      endif
7     continue
      do 80 i=1,nsize
```

```
      do 80 j=1,nsize
      h(i,j)=g(i,j)
 80   continue
c  template b4
      do 8 i=1,nsize
      do 8 j=1,nsize
      if(h(i,j).eq.1) goto 81
      goto 8
 81   p(2)=h(i,j+1)
      p(3)=h(i-1,j+1)
      p(4)=h(i-1,j)
      p(5)=h(i-1,j-1)
      p(6)=h(i,j-1)
      p(7)=h(i+1,j-1)
      p(8)=h(i+1,j)
      p(9)=h(i+1,j+1)
      k1=p(5)+p(6)+p(7)
      k2=p(2)+p(9)
      if(k1.eq.0.and.k2.eq.2) then
      g(i,j)=0
      endif
 8    continue
      do 90 i=1,nsize
      do 90 j=1,nsize
      h(i,j)=g(i,j)
 90   continue
 101  continue
      do 100 i=1,nsize
      do 100 j=1,nsize
      if(g(i,j).eq.1) then
      g(i,j)=0
      else
      g(i,j)=255
      endif
 100  continue
c Process the dots alone
      write(*,*) ' enter kk for dot'
      read(*,*) kk
      do 311 k=1,kk
      do 330 i=1,128
      do 330 j=1,128
```

```
      if((i.eq.mag(1).and.j.eq.hag(1)).or.(i.eq.mag(2).and.
     * j.eq.hag(2)).or.(i.eq.mag(3).and.j.eq.hag(3))) then
      d(i,j)=3
      else
      if(d(i,j).eq.0.and.d(i,j+1).eq.1) goto 340
      goto 331
340   if(d(i,j+2).eq.0) goto 330
      d(i,j+1)=2
      goto 330
331   if(d(i,j).eq.1.and.d(i,j+1).eq.0) goto 350
      goto 330
350   if(d(i,j-1).eq.0) goto 330
      d(i,j)=2
      endif
330   continue
      do 3100 j=1,128
      do 3100 i=1,128
      if(d(i,j).eq.0.and.d(i+1,j).eq.1) goto 3110
      goto 3120
3110  if(d(i+2,j).eq.0) goto 3100
      d(i+1,j)=2
      goto 3100
3120  if(d(i,j).eq.1.and.d(i+1,j).eq.0) goto 3130
      goto 3100
3130  if(d(i-1,j).eq.0) goto 3100
      d(i,j)=2
3100  continue
311   continue
      do 999 i=1,128
      do 999 j=1,128
      if(d(i,j).eq.3) then
      g(i,j)=0
      endif
999   continue
      do 110 i=1,nsize
      do 110 j=1,nsize
      img(i,j)=char(g(i,j))
110   continue
      write(2) ((img(i,j),j=1,nsize),i=1,nsize)
      close(2)
      stop
      end
```

```
c*****************************************************************
c  filename : dmod.for                                           *
c Description : In d.for and d1.for ( Lee's method ) we         *
c               still have a discontinuity in many              *
c               characters in which the linking algorithm       *
c               fails to do it . To minimise this problem       *
c               we insert the following two conditions :        *
c                  1. 2 < B(p1) < 6                             *
c                  2.     A(p1) = 1                             *
c*****************************************************************
      integer h(128,128),g(128,128),d2(128,128),d3(128,128)
      integer t(8),tm(8),q(8),tt(8),m,k,or,m1,m2,dd
      character img(128,128)
      character*16 filnme,filename
      write(*,*) 'enter the input filename'
      read(*,'(a16)') filnme
      write(*,*) 'enter the output filename'
      read(*,'(a16)') filename
      open(1,file=filnme,form='binary',status='old')
      open(2,file=filename,form='binary',status='new')
      do 1 i=1,128
      do 1 j=1,128
      read(1) img(i,j)
      h(i,j)=ichar(img(i,j))
1     continue
      close(1)
      do 2 i=1,128
      do 2 j=1,128
      if(h(i,j).eq.0) then
      h(i,j)=1
      else
      h(i,j)=0
      endif
2     continue
      do 3 i=1,128
      do 3 j=1,128
      if(h(i,j).eq.1) then
      do 4 k=1,128
      if(h(i,j).ne.h(i-k,j).or.h(i,j).ne.h(i+k,j)) then
      max1=k
      goto 5
```

```
            endif
 4          continue
 5          continue
            do 41 m=1,128
            if(h(i,j).ne.h(i,j-m).or.h(i,j).ne.h(i,j+m)) then
            max2=m
            goto 42
            endif
 41         continue
 42         continue
            if(max1.lt.max2) then
            g(i,j)=max1
            else
            g(i,j)=max2
            endif
            endif
 3          continue
c the new selection rules
            do 43 i=1,128
            do 43 j=1,128
            if(g(i,j)+1.gt.g(i,j-1).and.g(i,j)+1.gt.g(i-1,j))
           *  goto 45
            goto 44
 45         if(g(i,j)+1.gt.g(i,j+1).and.g(i,j)+1.gt.g(i+1,j))
           *  goto 46
            goto 44
 46         h(i,j)=g(i,j)
            goto 43
 44         h(i,j)=0
 43         continue
c linking algorithm
            do 123 i=5,128
            do 21 j=2,127
            if(d2(i,j).ne.0) then
c SUBPROGRAM A :
            if((d2(i-1,j-1).ne.0.and.d2(i+1,j+1).ne.0).or.
           *  (d2(i,j-1).ne.0.and.d2(i,j+1).ne.0).or.
           *  (d2(i+1,j-1).ne.0.and.d2(i-1,j+1).ne.0).or.
           *  (d2(i-1,j).ne.0.and.d2(i+1,j).ne.0).or.
           *  (d2(i-1,j-1).ne.0.and.d2(i-1,j+1).ne.0).or.
           *  (d2(i-1,j+1).ne.0.and.d2(i+1,j+1).ne.0).or.
```

```
     *  (d2(i+1,j-1).ne.0.and.d2(i+1,j+1).ne.0).or.
     *  (d2(i-1,j-1).ne.0.and.d2(i+1,j-1).ne.0)) then
        lt=1
        else
        lt=0
        endif
        if(lt.eq.0) then
c SUBPROGRAM B :
c n region
       * if(d2(i-1,j).eq.0) then
          if(d2(i-2,j).ne.0.or.d2(i-3,j).ne.0) then
            d3(i-1,j)=g(i-1,j)
             if(d2(i-3,j).ne.0) goto 21
          endif
         endif
c ne region
        if(d2(i-1,j).eq.0.and.d2(i-1,j+1).eq.0.and.
     *  d2(i,j+1).eq.0) then
        if(d2(i-3,j+1).ne.0.or.d2(i-3,j+2).ne.0.or.
     *  d2(i-3,j+3).ne.0.or.d2(i-2,j+3).ne.0.or.
     *  d2(i-1,j+3).ne.0) then
          d3(i-1,j+1)=g(i-1,j+1)
          goto 21
        endif
        endif
c e region
        if(d2(i,j+1).eq.0) then
          if(d2(i,j+2).ne.0.or.d2(i,j+3).ne.0.) then
           d3(i,j+1)=g(i,j+1)
        if(d2(i,j+3).ne.0) goto 21
          endif
       endif
c se region
        if(d2(i,j+1).eq.0.and.d2(i+1,j+1).eq.0.and.d2(i+1,j).
     *  eq.0.and.d2(i+2,j).eq.0.and.d2(i+3,j).eq.0) then
       if(d2(i+1,j+3).ne.0.or.d2(i+2,j+3).ne.0.or.d2(i+3,j+3).
     *  ne.0.or.d2(i+3,j+2).ne.0.or.d2(i+3,j+1).ne.0) then
        d3(i+1,j+1)=g(i+1,j+1)
        goto 21
        endif
        endif
c s region
```

```
      if(d2(i+1,j+1).eq.0.and.d2(i+1,j).eq.0) then
        if(d2(i+3,j).ne.0) then
         d3(i+1,j)=g(i+1,j )
        endif
      endif
      endif
      endif
 21    continue
 123    continue
c       thining
       do 7 i=1,128
       do 7 j=1,128
       g(i,j)=d2(i,j)+d3(i,j)
       if(g(i,j).eq.0) then
       h(i,j)=0
       else
       h(i,j)=1
       endif
 7     continue
       write(*,*)'enter dd'
       read(*,*) dd
       do 99 kk=1,dd
       do 20 i=1,128
       do 20 j=1,128
       if(h(i,j).eq.1) goto 100
       goto 20
 100   t(1)=h(i-1,j)
       t(2)=h(i-1,j+1)
       t(3)=h(i,j+1)
       t(4)=h(i+1,j+1)
       t(5)=h(i+1,j)
       t(6)=h(i+1,j-1)
       t(7)=h(i,j-1)
       t(8)=h(i-1,j-1)
       m1=t(1)+t(2)+t(3)+t(4)+t(5)+t(6)+t(7)+t(8)
       if(m1.le.6.and.m1.ge.2) goto 23
       goto 20
 23    continue
       init=0
       do 24 mm=1,7
       if(t(mm).eq.0.and.t(mm+1).eq.1) then
```

```
        init=init+1
        endif
 24     continue
        if(t(8).eq.0.and.t(1).eq.1) then
        init=init+1
        endif
        if(init.eq.1) goto 25
        goto 20
 25     continue
c tm
        tm(1)=t(1)
        tm(2)=0
        tm(3)=t(3)
        tm(4)=0
        tm(5)=t(5)
        tm(6)=0
        tm(7)=t(7)
        tm(8)=0
c Shift tm
        q(1)=0
        q(2)=tm(1)
        q(3)=tm(2)
        q(4)=tm(3)
        q(5)=tm(4)
        q(6)=tm(5)
        q(7)=tm(6)
        q(8)=tm(7)
c OR_ing t and q
        do 200 k=1,8
        or=t(k)+q(k)
        if(or.eq.0) then
        tt(k)=0
        else
        tt(k)=1
        endif
 200    continue
        n1tt=tt(1)+tt(2)+tt(3)+tt(4)+tt(5)+tt(6)+tt(7)+tt(8)
        n10tt=0
        do 300 m=2,8
        if(tt(m-1).eq.1.and.tt(m).eq.0) then
        n10tt=n10tt+1
        endif
```

```
300     continue
        if(tt(8).eq.1.and.tt(1).eq.0) then
        n10tt=n10tt+1
        endif
        if(n10tt.gt.1) goto 1000
        goto 1001
1000    h(i,j)=1
1001    if(n10tt.eq.1) goto 1002
        goto 20
1002    if(n1tt.eq.1) then
        h(i,j)=1
        else
        h(i,j)=0
        endif
20      continue
        do 40 i=1,128
        do 40 j=1,128
        if(h(i,j).eq.0) then
        h(i,j)=255
        else
        h(i,j)=0
        endif
40      continue
        do 50 i=1,128
        do 50 j=1,128
        img(i,j)=char(h(i,j))
50      continue
        write(2) ((img(i,j),j=1,128),i=1,128)
        close(2)
        stop
        end
```

```
c*************************************************************************
c   filename : thick.for                                                 *
c                                                                        *
c     Description : This program is implemented to perform              *
c                   thickening algorithm using scanning                 *
c                   method                                              *
c                                                                        *
c*************************************************************************
c
      integer h(128,128)
      character img(128,128),imh(128,128)
      integer kk
      character*16 filnme
      write(*,*) ' enter the filename '
      read(*,'(a16)') filnme
      open(1,file=filnme,form='binary',status='old')
      open(2,file='out.img',form='binary',status='new')
      do 20 i=1,128
      do 20 j=1,128
      read(1) img(i,j)
      h(i,j)=ichar(img(i,j))
      if(h(i,j).eq.0)then
      h(i,j)=1
      else
      h(i,j)=0
      endif
20    continue
      write(*,*) ' enter kk'
      read(*,*) kk
      do 11 k=1,kk
      do 30 i=1,128
      do 30 j=1,128
      if(h(i,j).eq.0.and.h(i,j+1).eq.1) goto 40
      goto 31
40    h(i,j)=2
      goto 30
31    if(h(i,j).eq.1.and.h(i,j+1).eq.0) goto 50
      goto 30
50    h(i,j+1)=2
30    continue
```

```
      do 100 j=1,128
      do 100 i=1,128
      if(h(i,j).eq.0.and.h(i+1,j).eq.1) goto 110
      goto 120
110   h(i,j)=2
      goto 100
120   if(h(i,j).eq.1.and.h(i+1,j).eq.0) goto 130
      goto 100
130   h(i+1,j)=2
100   continue
      do 60 i=1,128
      do 60 j=1,128
      if(h(i,j).eq.0) then
      h(i,j)=0
      else
      h(i,j)=1
      endif
60    continue
11    continue
      do 80 i=1,128
      do 80 j=1,128
      if(h(i,j).eq.1) then
      h(i,j)=0
      else
      h(i,j)=255
      endif
80    continue
      do 140 i=1,128
      do 140 j=1,128
      imh(i,j)=char(h(i,j))
140   continue
      write(2) ((imh(i,j),j=1,128),i=1,128)
      close(2)
      stop
      end
```

VITA AUCTORIS

Mohamed TELLACHE

September 1960	Born in Draa-El-mizan (ALGERIA)
June 1980	Completed High School (Baccalaureat) Lycee Abane Ramdane (Algiers) ALGERIA
June 1985	Graduated in Electronics (Ingenieur) Ecole polytechnique d'Alger, ALGERIA
September 1986	Admitted in the Master's program, Department of Electrical Engineering University of Windsor
June 1988	Candidate for the degree of M.A.Sc. Electrical Engineering University of Windsor, Ontario, CANADA N9b-3P4