



National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

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.

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

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.

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

**A Primitive Selection Method
For
Unconstrained Line Structures**

Siamak Moradmand

A Major Technical Report
in
the Department
of
Computer Science

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Computer Science at
Concordia University
Montréal, Québec, Canada

June 1988

© Siamak Moradmand, 1988

Permission has been granted to the National Library of Canada to microfilm this thesis and to lend or sell copies of the film.

The author (copyright owner) has reserved other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without his/her written permission.

L'autorisation a été accordée à la Bibliothèque nationale du Canada de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur (titulaire du droit d'auteur) se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation écrite.

ISBN 0-315-51360-8

ABSTRACT**A Primitive Selection Method
For
Unconstrained Line Structures****Siamak Moradmand**

In syntactic (structural) pattern recognition, each pattern is expressed as a composition of its constituents called subpatterns or pattern primitives or atoms. Evidently, for this approach to be advantageous, these atoms should be much easier to recognize than the patterns themselves. Decomposition (fragmentation) algorithms partition a pattern into simple parts. The need for such algorithms exists because global features are not suitable for describing complex patterns.

A decomposition algorithm for unconstrained (i.e. Chinese characters) line structures is proposed which fragments the structure into primitives. Each primitive is represented by a junction point and the segments connecting to it. We will verify which part of the segment (skeleton, contour or both) is the best representative of its nature and most reliable in such a representation scheme.

ACKNOWLEDGEMENTS

I wish to express my sincere gratitude to my advisor Dr. Tony Kasvand, for his expert guidance throughout the course of this research. He has always been accessible for discussions, suggestions and has been a major source of motivation during the completion of this report.

I would also like to gratefully acknowledge the influence of professor C. Y. Suen, on my interest in the field of Pattern Recognition.

I dedicate this report in memory of my beloved mother Razyieh Doostan.

Table of Contents

Abstract	iii
Acknowledgement	iv
List of Figures	v
List of Tables	vi
1. Introduction	1
1.1 Primitive Selection	3
2. Preprocessing	10
2.1 Distance to Closest Contour	10
2.2 Thinning	13
2.2.1 Constrained Distance Thinning	15
2.2.2 Properties of Skeleton	18
2.2.3 Correction of Thinning Defects	23
2.3 Skeleton Labelling	25
2.4 Segments labels dilation	28
3. Fragmentation Algorithm	31
3.1 (r, θ) Description	35
3.2 Normalization and Weighing	42
3.3 Matching	46
4. Atoms Interrelationship and Object Recognition	52
5. Results of Experiments	55
6. Conclusion	59
7. References	60
8. Appendix I	63

LIST OF FIGURES

Figure 1.1-1	A structural description of a Chinese character	9
Figure 2.1-1	Connectivity on a square grid	10
Figure 2.1-2(a)	The binary image, $Inpix(i,j)$	12
Figure 2.1-2(b)	Distanced image, $Distpix(i,j)$	12
Figure 2.2.1-1	Thinned image, $Thinpix(i,j)$	17
Figure 2.2.2-1	The basic properties of the perfectly thinned image	18
Figure 2.2.2-2	Some rather rare junction configuration	20
Figure 2.2.2-3	The damage done to 90 degree corners	20
Figure 2.2.2-4	Marked thinned image, $JSpix(i,j)$	22
Figure 2.2.3-1	Special thinning defects	23
Figure 2.2.3-2	Thinned image after correction of thinning defects	24
Figure 2.3-1(a)	Segment labelled image, $LJ-pix(i,j)$	27
Figure 2.3-2(b)	Labelled skeleton image, $LJSpix(i,j)$	27
Figure 2.4-1	Dilated segment labels image, $DILpix(i,j)$	30
Figure 3-1	Labelled contour of the image	34
Figure 3.1-1	A "contour" atom before and after chopping	38
Figure 3.1-2	"Contour" atoms	39
Figure 3.1-3	"Skeleton" atoms	40
Figure 3.1-4	"Contour and Skeleton" atoms	41
Figure 3.2-2	Normalized "contour" atom	44
Figure 3.2-3	Weighed "contour" atom	45
Figure 5-1	Ideal atoms	58

LIST OF TABLES

Table 3.3-1	Matching results of "contour" atoms	49
Table 3.3-2	Matching results of "skeleton" atoms	50
Table 3.3-3	Matching results of "contour and skeleton" atoms	51

1.- INTRODUCTION

The many mathematical techniques used to solve pattern recognition problems may be grouped into two general approaches [1,2]. They are the decision-theoretic or statistical approach and structural or syntactic approach [3,4]. In the decision-theoretic approach, a set of characteristic measurements, called features, are extracted from the pattern. Each pattern is represented by a feature vector, and the recognition of each pattern is usually made by partitioning the feature space. On the other hand, in the syntactic approach, each pattern is expressed as a composition of its constituents called, subpatterns or pattern primitives(atoms). This approach draws an analogy between the structure of pattern and the syntax of a language. The recognition of each pattern is usually made by parsing the pattern structure according to a given set of syntax rules. Patterns are specified as building out of pattern primitives in various ways of composition just as phrases and sentences are built up by concatenating words and words are built up by concatenating characters.

Evidently, for this approach to be advantageous, the simplest subpatterns selected, called "pattern primitives or atoms", should be much easier to recognize than the patterns themselves. The "language" which provides the structural description of patterns in terms of a set of pattern primitives and their composition operations, is called " pattern description language". The rules governing the composition of primitives into pattern are usually specified by the so-called "grammar" of the pattern description language. After each primitive within the pattern is identified, the recognition process is accomplished by performing a syntax analysis or parsing of the "sentence" describing the

given pattern to determine whether or not it is syntactically(or grammatically) correct with respect to the specified grammar.

A syntactic pattern recognition system can be considered as consisting of three major parts [7]; namely, preprocessing, pattern description or representation, and syntax analysis. The function of preprocessing include: (i) pattern encoding and approximation, and (ii) filtering, restoration and enhancement. An input pattern is first coded or approximated by some convenient form for further processing. For example, a black and white picture can be coded in terms of a grid(or a matrix) of 0's and 1's, or a waveform can be approximated by its time samples or a truncated Fourier's series expansion. In order to make the processing in the later stages of the system more efficient, some sort of "data compression" is often applied at this stage. Then, techniques of filtering, restoration and/or enhancement will be used to clean the noise, to restore the degradation,and/or to improve the quality of the coded(or approximated) patterns. At the output of preprocessing, presumably we have patterns with reasonably "good quality". Each preprocessed pattern is then represented by a language-like structure.

The operation of pattern-representation process consists of (i) pattern segmentation, and (ii) primitive(or atom) extraction. In order to represent a pattern in terms of its subpatterns, we must segmentize the pattern and, in the meantime, identify(or extract) the primitives and relations in it. In other words, each preprocessed pattern is segmentized into subpatterns and pattern primitives(atoms) based on prespecified syntactic or composition operations. Each pattern is now represented by a set of primitives with specified syntactic operations. For example, in terms of "concatenation" operation, each pattern is

represented by a string of (concatenated) primitives. More sophisticated systems should be able to detect various syntactic relations within the pattern.

The decision on whether or not the representation(pattern) is syntactically correct(i.e., belongs to the class of patterns described by the given syntax or grammar) will be performed by the "syntax analyzer" or "parser". When performing the syntax analysis or parsing, the analyzer can usually produce a complete syntactic description, in terms of a parse or parsing-tree, of the pattern, provided it is syntactically correct. Otherwise, the pattern is either rejected or analyzed on the basis of other given grammars, which presumably describe other possible classes of patterns under consideration.

1.1- PRIMITIVE SELECTION

The first step in formulating a syntactic model for pattern description is the determination of a set of primitives in terms of which the patterns of interest may be described. This will be largely influenced by the nature of the data, the specific application in question, and the technology available for implementing the system. There is no general solution for the primitive selection problem at this time[5,6]. The following requirements usually serve as a guideline for selecting pattern primitives[7] :

(i) The primitives should serve as basic pattern elements to provide a compact but adequate description of the data in terms of specified structural relations(e.g., the concatenation operation).

(ii) The primitives should be easily extracted or organized by existing non-linguistic methods, since they are considered to be simple and compact patterns and their structural information not important.

For example for speech patterns, phonemes are naturally considered as a good set of primitives with the concatenation relation. Similarly, strokes have been suggested as primitives in describing handwriting. However, for pictorial patterns, there is no such "universal picture element" analogous to phonemes in speech or strokes in handwriting.

Sometimes, in order to provide an adequate description of the patterns, the primitives should contain the information which is important to the specific application in question. For example if the size(or shape or location) is important in the recognition problem, then the primitives should contain information relating to size(or shape or location) so that patterns from different classes are distinguished by whatever method is to be applied to analyze the descriptions. This requirement often results in a need for semantic information in describing primitives[6].

One of the earliest papers describing the decomposition of pictorial patterns into primitives[8] presented a conceptually appealing method which allows the recognition system to (heuristically) determine the primitives by inspection of training samples. A pattern is first examined by a programmed scan, the result of the scan is to produce descriptions of segments of the picture which are divisions conveniently produced by the scanning process, and not necessarily true divisions. The scanning process also includes preprocessing routines for noise-cleaning, gap-filling, and curve-following. The subpictures obtained in the

scan are analyzed and connected, when appropriate, into true picture parts; a description is given in terms of the length and slope of straight-line segments and the length and curvature of curved segments. The structural relations among various segments(primitives) of a picture is expressed in terms of a connection table(table of joints). The assembly program produces a "statement" which gives a complete description of the pattern. The description is independent of the orientation and the size of the picture, the lengths of the various parts being given relative to one another. It is, in effect, a coded representation of the pattern and may be regarded as a one-dimensional string consisting of symbols chosen from a specified alphabet. The coded representation gives the length, slope and curvature to other of each primitive, together with details of the ends and joints to other primitives.

A formal model for the abstract description of english cursive script has been proposed by EDEN and HALLE[9]. The primitives are four distinct line segments in the form of a triplet:

$$\sigma_j = [(x_{j_1}, y_{j_1}), (x_{j_2}, y_{j_2}), \theta_j]$$

where (x_j, y_j) 's represent the approximate location of the end points of the line segment, and θ_j refers to the sense of rotation from the first to the second end point. θ_j is positive if the sense of rotation is clockwise and negative if counter clockwise. The four primitives are:

$$\sigma_1 = [(1,0), (0,0), +]$$

"bar"




$$\sigma_2 = [(1,1), (0,0), +]$$


"hook"



$$\sigma_3 = [(0,0), (0,1), +]$$

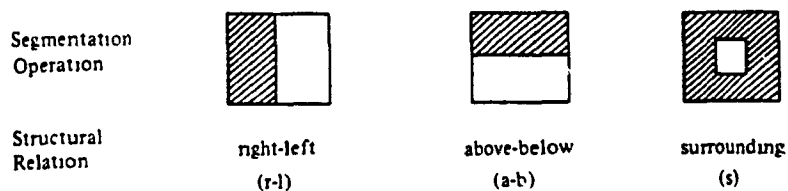
“arch” 

$$\sigma_4 = [(1,\epsilon), (0,0), +]$$

$0 < \epsilon < 1$, “loop” 

They can be transformed by changing the sign of θ or by reflection about the horizontal or vertical axis. These transformations generate 28 strokes (because of symmetry, the arch generates only four strokes), but only nine of them are of interest in the English script commonly used.

Another example is the recognition of Chinese characters [10,11,12,13], from the knowledge about the structure of Chinese characters. A small number of simple segmentation operations such as:



can be used. Each operation also generates a particular structural relation between the two neighbouring primitives. Applying these operations recursively, that is, segmentizing each subpattern again by any one of the three operations, we can segmentize a Chinese character into its primitives. If the primitives can be extracted or recognized by existing techniques, a Chinese character can be described syntactically with the given set of structural relations. An illustrative example is given in figure 1.1-1. It is anticipated that the resulting structural descriptions will be much more complex if we choose basic strokes as the primitives.

C. Y. Suen[14] has classified the structural features into the following main families:

- Line segments and edges: Edges and line segments are detected from the character. From the above information, such features as line lengths, and line ends can be obtained.
- Outline of the character: The contour of a character is traced. Contour tracing can generate the following features:
 - Line tips or end points.
 - Length of line segments, including perimeter.
 - Sharp angles, protrusions and spikes: $>$ \wedge λ
 - Arcs, bends: \cap \subset
 - Splits: $--$
 - Loops, circles: \bigcirc
 - Points of inflection: \int
 - Concavities and convexities: \curvearrowright \curvearrowleft
- Center-line of the character: By applying the thinning process on a character outline, the center-line of a character commonly called skeleton is obtained. From the skeleton the following features can be obtained:
 - Line tips.
 - Straight line segments, horizontal and vertical.
 - Diagonal lines, slants.
 - Concavities.
 - Loops.
 - Intersections, forks, branches, nodes.

More general methods for primitive selection may be grouped roughly into methods emphasizing boundaries(or skeleton)[15,16] and methods emphasizing regions[17,18]. For line patterns or patterns described by boundaries or skeletons, line segments are often suggested as primitives. A straight line segment could be characterized by the locations of its beginning(head) and end(tail), its length, and/or slope. Similarly, a curve segment might be described in terms of its head and tail and its curvature. The information characterizing the primitives can be considered as their associated semantic information or as features used for primitive recognition. Through the structural description and the semantic specification of a pattern, the semantic information associated with its subpatterns or the pattern itself can then be

determined. For pattern description in terms of regions, half-planes have been proposed as primitives. Shape and texture measurements are often used for the description of regions.

In the remainder of this report we propose a primitive selection method for images of unconstrained line structures. Maps, sketches, wiring diagrams, engineering drawings, musical notes and pentagrams, pages of printed or hand written text, etc., are all examples of images which contain structures of lines. The interpretation or the meaning of the configuration of lines in such images depends on their origin. If the image is a line drawing of a natural scene, then it is immediately understandable to us. However, if the image represents, say some Chinese writing, or a logic circuit diagram, then specific knowledge(learning) is required before the contents can be understood.

There is no reason to believe that the image processing mechanism differs according to whether we are reading a page of text, studying a circuit diagram, or we are trying to recognize some ideograms. Of course, recognition of complex line configuration is a closed loop process, where the "knowledge base" determines which configuration of lines are meaningful and which are not, but before use of such knowledge can be made, the lines in the image have to be described in a convenient mathematical or computational form.

For maximum generality, it will be assumed that the image content is completely unknown. The line structures in the image may be in any positions and orientations, and they may overlap. There are no restrictions on the number of line structure in the image, nor on the size of the structures. The only constraints on the image content are: (i) image is in binary form, (ii)

the lines in the binary image can be thinned without loss of significant information or seriously degrading the original information. A relatively lengthy preprocessing [29] sequence is required to bring information in the image into a form suitable for decomposition(fragmentation) algorithm.

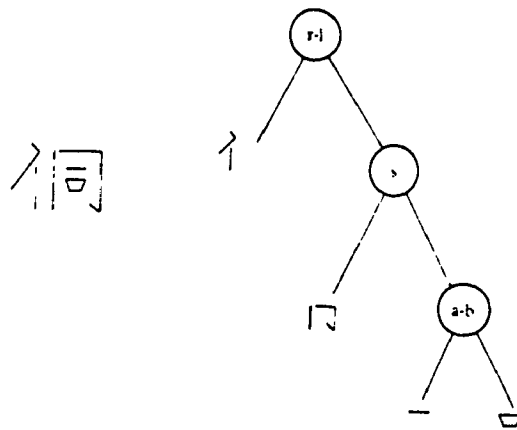


Figure 1.1-1: A structural description of a Chinese character.

2.- PRE-PROCESSING

Preprocessing is necessary to modify the input such that it meets the requirements of the fragmentation algorithm. The preprocessing includes of: (i) Distance Function, (ii) Thinning, (iii) Labelling and (iv) Dilatation of Segment Labels.

2.1- DISTANCE TO CLOSEST CONTOUR

In conventional usage, some form of a distance measure is computed over all the 1-valued pixels within a "blob" to the closest contour pixel of the blob. This gives a measure of the object. Thus, the conventional distance calculations give a distance function $\text{Dist}(i,j)$ value for each of the 1-valued pixels in the image. $\text{Dist}(i,j)$ is the distance of pixel (i,j) to the contour pixel (i_c, j_c) which is closest to it.

The algorithm uses the 3 by 3 neighborhood N , they require a definition of connectivity within N (4- or 8- connected), and are iterated until all the pixels within all the blobs have been assigned a distance value. The connectivity patterns are shown in figure 2.1-1.

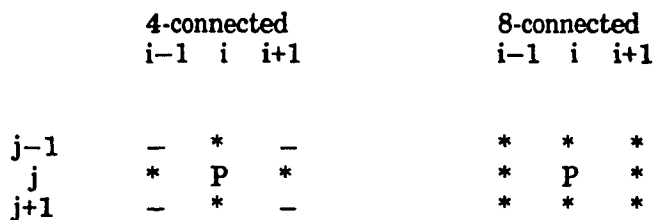


Figure 2.1-1: Connectivity on a square grid within a 3 by 3 neighbourhood N with the central pixel P . The peripheral pixels N' are indicated by *'s. Pixels indicated by -'s are not used. (a) 4-connectivity. (b) 8-connectivity.

In the distance computations, however, the distance value is loaded back into image to serve as a memory of distance already computed, since the range of the 3 by 3 operator is only one pixel. The background pixels in the image are indicated by zeros (0's) and the foreground pixels are indicated by one's (1's). Thus, it is simplest to assign the value 2 to contour pixels, the value 3 to the pixels next to the 2-valued pixels (contour), 4 to those next to 3, etc. The procedure stops when all the 1-valued pixels "have been used up". The input image is $Inp_{i,j}$ and the resultant distance image is $Dist_{i,j}$, See figure 2.1-2.

The "distance to the closest contour" function gives the distance of every 1-valued pixel to its closest contour. Likewise if the distance is computed over the background (0-valued) pixels, then the distance of these pixels to their respective closest contours are obtained. Consequently the distance function can be used to measure the local thickness of the objects in the foreground and the background distances (gaps) between the objects.

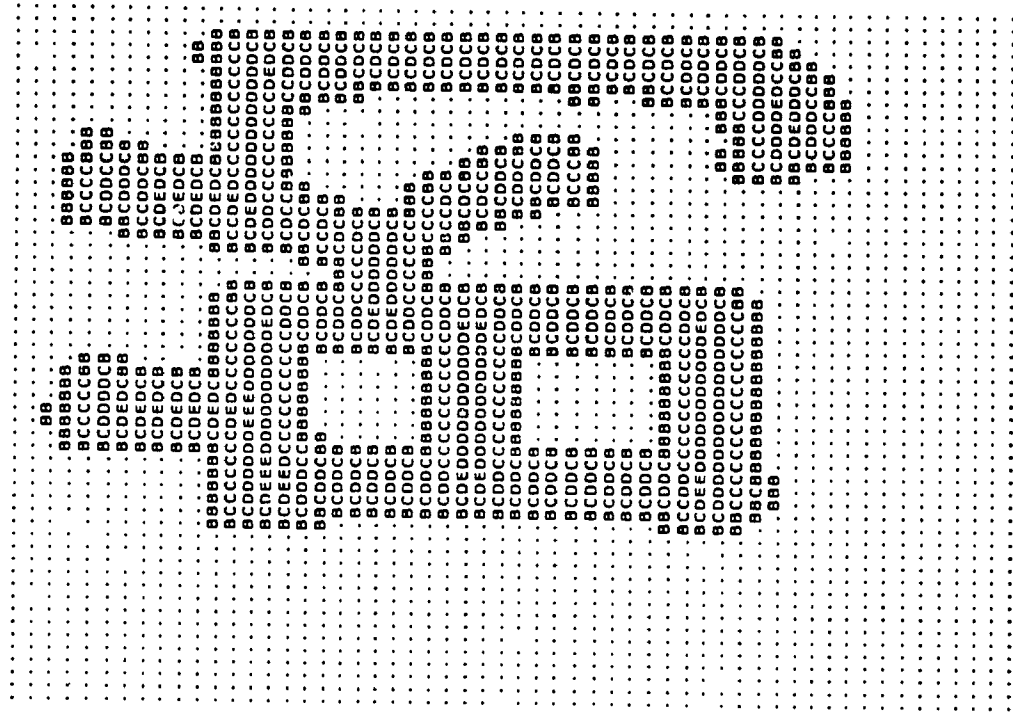


Figure 2.1-2(a): The binary image, Impix(i,j).

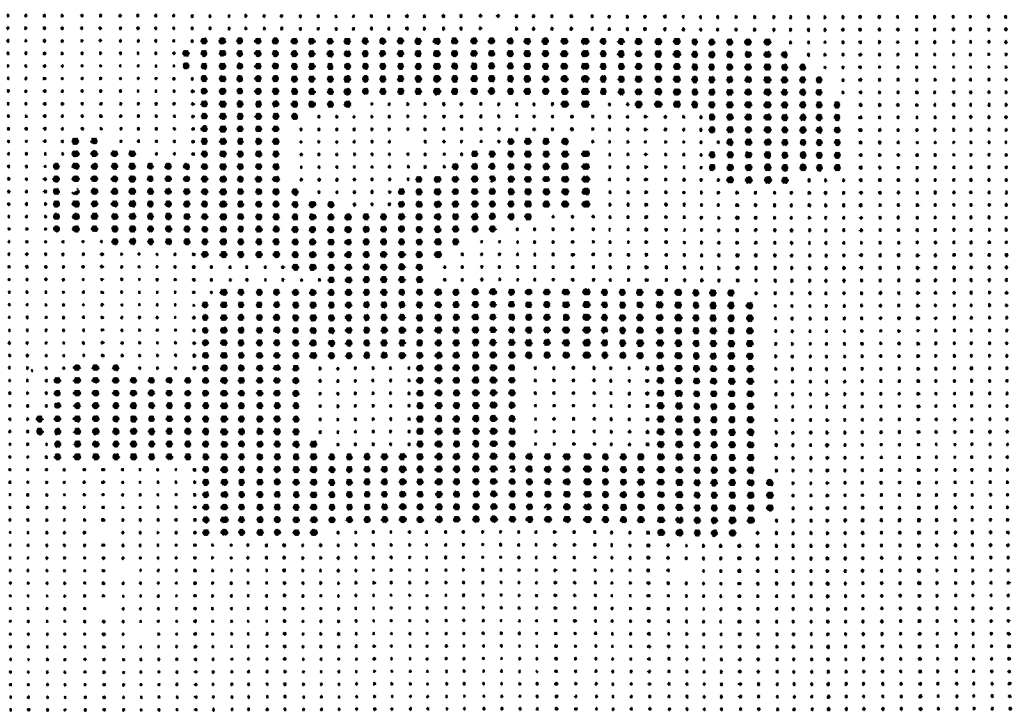


Figure 2.1-2(b):

The distanced image, Distpix(I,J). The sequence numbers (2,3,4,...) are indicated by upper case letters (B,C,D,...).

2.2- THINNING

The so-called skeletonization or thinning of an object may be said to "remove the flesh of the object and only leave its skeleton". For example, a four-cornered star or diamond would be represented by two crossing thin lines, a thick binary image becomes a thin binary line in the middle of the thick line, a circle reduces to a point at its center, an ellipse reduces to its major axes, and so on. Ideally, thinning reduces a thick binary object to its skeleton which consists of thin binary lines of connected pixels, i.e., there are no gaps in the lines.

In principle, the result of thinning should resemble what we would ourselves draw as the basic axes, the backbone, the skeleton, or the "stick figure" of an object. However, in practice we should not expect such ideal results since our concepts of what constitutes a skeleton of an object is based on different knowledge than what is available to the computer at this stage of processing. Thinning produces results which may resemble the ideal skeleton of an object only in very idealized circumstances since thinning is very sensitive to noise.

There are more than a dozen different thinning methods, depending on the form of the skeleton that one wants or rather hopes to obtain. The results of thinning also depend on the connectivity used, i.e., whether the final skeleton is to be 4-connected or 8-connected.

Among the many methods, only the so-called "perfect thinning" is used here [20]. The word "perfect" does not mean that the thinning is perfect, actually the result could be very poor indeed. "Perfect" in the present case

only means that no more pixels can be removed from the skeleton, except from the ends of the thin lines, without breaking the connectivity of the skeleton (thin lines). Thus if the ends of already thin lines and single pixels are not to be removed, there is not even one single pixel that can be removed without breaking the connectivity of the skeleton. This is absolutely true for perfect thinning and that is where the name "perfect" originated.

Thinning is an iterative operation which removes edge pixels from the object. One "layer" of pixel is removed per iteration cycle. However, if the pixel to be removed is going to break connectivity of the remaining pixels, then this pixel is not removed. The iteration stops when no more pixels can be removed. Thinning may be looked upon as a form of constrained erosion where the erosion does not remove a pixel if the removal would break connectivity between the remaining pixels. Pixel removal is also stopped at the ends of already thin lines, otherwise a thin line would be "chewed away" at the ends until only one pixel remains.

Our visual system presumably uses a closed loop process where information from the recognition stages feeds back to preprocessing allowing the alternatives provided by preprocessing to be used selectively or where the poor results are suppressed. Of course, such a feedback has to be done also in computer vision systems, but we have to clearly acknowledge at what stage of the processing we are at the moment and what information is available.

There are several possible ways to constrain the thinning process, depending on the situation and what one wants to achieve. Frequently, the different methods are used in various combinations:

- 1) Try to remove the "sources of trouble" before the thinning process is started.
- 2) Use some measure of "local importance" of the region or some specific logic to control the thinning.
- 3) Synchronize the thinning process between two or more images such that the results are in the registration.
- 4) Constrain the thinning process to specific labelled objects rather than applying it to all the 1-valued pixels indiscriminantly.
- 5) Correct the results after thinning.

Among the constraints we chose "Constrained distance" which works as following.

2.2.1- CONSTRAINED DISTANCE

The 8-connected 3 by 3 neighbourhood N8 is used, with its central pixel P and periphery N8'. Basically the problem is to constrain the thinning process such that one image is the constraint(Distpix(i,j)) while another one(Inpix(i,j)) is eroded subject to the constraint of not breaking connectivity. From the N8 neighbourhood it is impossible to decide if a single 0-valued pixel represents a hole in an object or if the 0-valued pixel is at the periphery of the object. Likewise, a 1-valued pixel on a spur cannot be distinguished from an end pixel on a thin or a thinned line. Thus, the necessary information to "tame" the thinning process is not contained within the local 3 by 3 neighbourhood N8 which is used by the thinning logic. It may be noted that even the thinning operation itself requires a slightly larger "view" than N8 since track has to be kept of the pixels removed during thinning. In order to constrain the thinning process, there are several questions that need clear answers, remembering that the objects are unknown. The main questions are:

- a) Where do we want the skeleton to be with respect to the object?

b) What information is or can be made available for constraining thinning?

The simplest answer to both questions, in the absence of knowledge about object identity, is to force the skeleton to be as close as possible to the local maxima, ridge, and other central points of the distance to the closest contour function. This places the skeleton pixels at equal distance from the neighbouring closest contours. The procedure involves "hill climbing" on the distance function(measure), erosion constrained not to break connectivity, and thinning. However, the elementary distance function is very sensitive to noise within the objects but is not sensitive to spurs on the contours of the objects. Thus, the elementary distance function is suitable for controlling spurs on the objects but is not suitable for controlling the effects of holes within the objects. However, the distance function can also be constrained, for example, to compute the distance to n ($n=2,3,\dots$) closest contour pixels instead of just to one ($n=1$) contour pixel. Figure 2.2.1-1 shows the result of thinning.

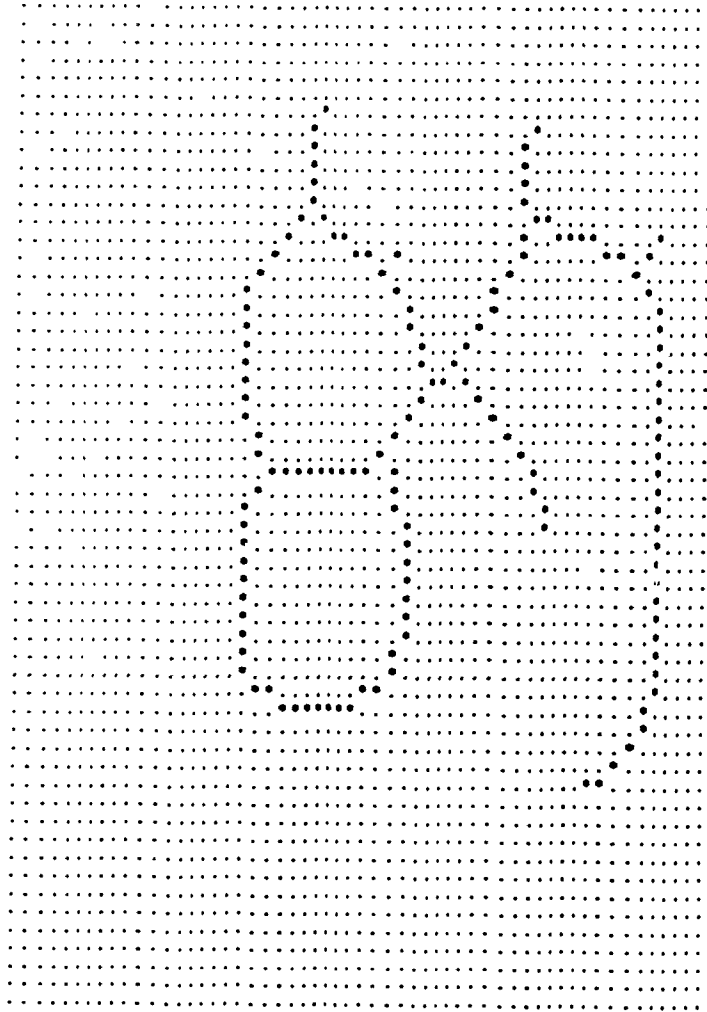


Figure 2.2.1-1: Thinned image, $\text{Thinpix}(i,j)$.

2.2.2. PROPERTIES OF PERFECTLY THINNED SKELETONS

Even though elementary thinning is very sensitive to noise within the image and to "spurs" or "fur" at the edges of objects, and also rather erratic and time consuming to carry out, the perfectly thinned skeleton possesses many important and useful properties which are rather difficult to find by other means.

The properties of the perfectly thinned skeletons can be detected by very simple logic based on the pixels within the 8-connected 3 by 3 neighborhood N . The various cases are illustrated in figure 2.2.2-1.

0 0 0	0 0 0	0 0 1	1 0 1	1 0 1	1 0 0	* * *	* * *
0 1 0	0 1 0	0 1 0	0 1 0	0 1 0	0 1 1	* P *	* * *
0 0 0	0 1 0	0 1 0	0 1 0	1 0 1	1 1 0	* * *	* * *
(a)	(b)	(c)	(d)	(e)	(f)	(N)	(N')

Figure 2.2.2-1: The basic properties of the perfectly thinned skeletons. The 1-valued pixel at P is : (a) A lonely pixel. (b) A line end. (c) Pixel on a line segment. (d,e and f) Junction pixels. (N' and N) The 3 by 3 neighbourhood.

Clearly, if the central pixel P of N is 1-valued, and we count the number n of 1-valued pixels in N' , then:

- 1) If $n=0$ then the pixel P may represent just a single noise pixel or it may be the centre of some more or less circular object(case (a)).
- 2) If $n=1$ then P is some form of "end" pixel (case (b)). Whether this "end" belongs to a long line or to a very short line, or is even a poorly formed centre of a "blob", we do not know yet. Several further computations have to be carried out before this decision can be made.

- 3) If $n=2$ then P is a pixel on a line segment (case (c)). Again, whether this line segment is long or short, or even if we want to call it a line, is not yet known.
- 4) If $n>2$ then P is some form of "junction" pixel (cases (d), (e) and (f)). In other words some form of "split" is occurring in the object, or two or more lines may be crossing. All we know at this moment is that this region could not be thinned down to any of the forms shown in cases (a), (b), or (c).

The above information about the pixel P is obtained with the following elementary logic: If (P.eq.1) then count pixels in N' . Use the count n as described. Thus, if the input image is called $\text{Thinpix}(i,j)$, then

```

For all pixels j and i Do
  If(thinpix(i,j).eq.1) Then
    Count 1-valued pixels in  $N'$ 
  Enddo i and j

```

The different pixel types in the perfectly thinned image will be considered to belong to two categories, namely, the junction pixels (case 4), and all the "other" pixels (cases 1, 2, and 3). The junction pixels indicate that some form of line crossing is taking place. The "other" pixels will be called "segment" pixels from now on, since they imply some form of line in the image. At this stage of processing we still do not know which pixels can be considered to "belong together" and form larger entities.

The junction pixels tend to have many different configurations, depending on the complexity of the junction regions in the image. If the image has very poor spatial resolution, then very large junction region may be formed. This situation is easy to detect after some further processing. However, there are two special situation that sometimes occur, which should be known. These are shown in figure 2.2.2-2, where the junction is a square block of four pixels, or

a diamond of five pixels. The centre of the diamond region may be set to zero, since the crossing number is eight. These configurations do not have any special significance.

0 0 0 0	0 0 1 0 0	0 0 1 0 0
1 0 0 1	0 0 1 0 0	0 0 1 0 0
0 1 1 0	1 1 1 1 1	1 1 0 1 1
0 1 1 0	0 0 1 0 0	0 0 1 0 0
1 0 0 1	0 0 1 0 0	0 0 1 0 0

Figure 2.2.2-2: Some rather rare junction configurations.

In addition to all the other difficulties with thinning, the further “damage” done by perfect thinning is best seen at 90 degree corners that are lined up with the square grid. A real case is shown in figure 2.2.2-3. This kind of behaviour is not desirable, since perfect square corners have been removed. The situation, however, is rather unusual since it is seldom possible to line up an image so exactly with the quantizing grid that horizontal and vertical lines remain as single rows or columns of pixel.

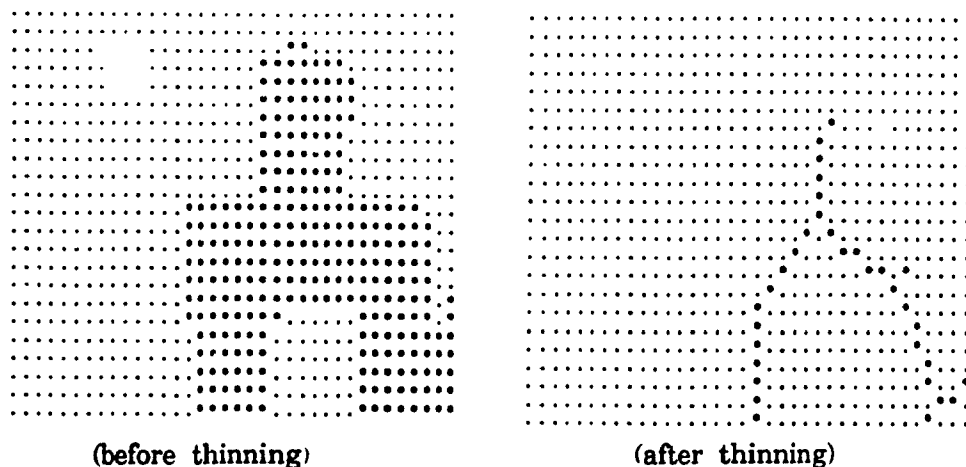


Figure 2.2.2-3: The “damage” done to 90 degree corners.

The next processing steps consist of separating the junction pixels from the segment pixels. One may create two new images, call them Junction-pix(i,j) and Segment-pix(i,j). The original binary input image is called Inpix(i,j) and the perfectly thinned image is called Thinpix(i,j). The thinned image was obtained from the function "Thin", i.e.:

$$\text{Thinpix}(i,j) = \text{Thin}(\text{Inpix}(i,j), \text{connect}=8, \text{Pixel-type}=1, \text{Max-iter}=\text{sufficient})$$

In order to separate the junction and segment pixels, the procedure consists of the following "program" steps:

```

For all pixels j and i Do
  Junction-pix(i,j)=0
  Segment-pix(i,j)=0
  If(Thinpix(i,j).eq.1)Then
    compute n ; (number of 1-valued pixels in N')
    If (n > 2)Then
      Junction-pix(i,j) = 1
      Segment-pix(i,j) = 0
    Else
      Junction-pix(i,j) = 0
      Segment-pix(i,j) = 1
    Endif
  Endif
Enddo i and j

```

For descriptive convenience, this "program" can be separated into two operations, where one computes the junctions only, and the other computes the segments only. Thus, for notational convenience, these operations will be written as:

$$\text{Junction-pix}(i,j) = \text{Junctions}(\text{Thinpix}(i,j)) \text{ and}$$

$$\text{Segment-pix}(i,j) = \text{Segments}(\text{Thinpix}(i,j)).$$

In order to save computer memory space, the junction and end point pixels may be marked negative (-1), and the segment pixels positive (+1). The background pixels in both cases are 0 (zero). Now only one output image is needed. Call it JSpix(i,j) where the J represents "junctions" and S represent "segments", see figure 2.2.2-4.

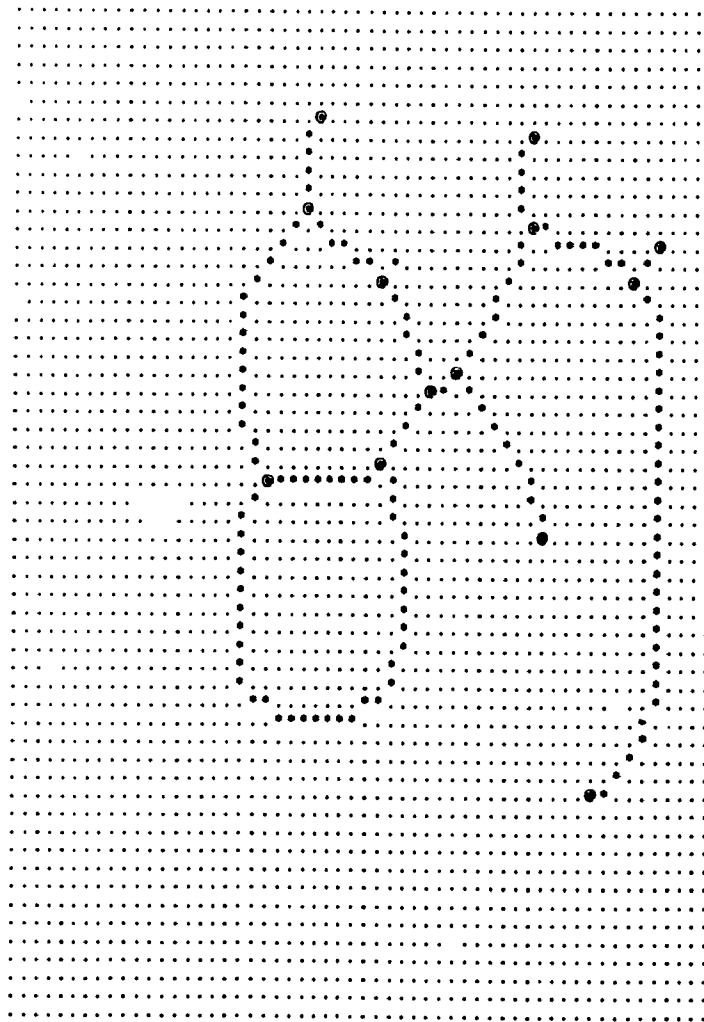


Figure 2.2.2-4: Marked thinned image, junctions and end points are indicated by @'s and segments are indicated by *'s.

2.2.3- CORRECTION OF THINNING DEFECTS

The perfectly thinned binary image $\text{Thinpix}(i,j)$ is obtained from the input binary image $\text{Inpix}(i,j)$ by thinning function called "Thin". The details and the properties of the "perfectly" thinned image were given in section 2.2.2. Symbolically, this process was indicated by:

$$\text{Thinpix}(i,j) = \text{Thin}(\text{Inpix}(i,j), \text{connectivity}=8, \text{Pixel-type}=1)$$

There are three special cases that may cause some problems in later analysis based on the perfectly thinned images. One is the perfectly closed thin line which has neither an end nor a beginning, see figure 2.2.3-1(a). The other case is the doubly connected segment which is only one pixel long but joins two junction pixels (or a junction and an end point), see figure 2.2.3-1(b). The last but not least is the case of a very long segment which could be broken into two new segments, see figure 2.2.3-1(c).

00000000000000000000	00000000000000000000	0x0000000000000000x0
00000000000000000000	00000000000000000000	0100000000000000010
00000000000000000000	00000000000000000000	0010000000000000100
00000000000000000000	00000000000000000000	00010000000000001000
00000111111000000000	00000010000010000000	0000100000000010000
00001000000100000000	00000001000100000000	00000100000000100000
00001000000100000000	00000000x1x000000000	00000010000001000000
00000100001000000000	00000001000000000000	00000010000001000000
00000011110000000000	00000010000000000000	00000001111100000000
00000000000000000000	00000000000000000000	00000000000000000000
(a)	(b)	(c)

Figure 2.2.3-1: Special cases. (a) The perfectly closed thin line with no junctions and ends. (b) The doubly connected single pixel long segment indicated by the underlined 1-valued pixel. (c) The very long segment. The x's represent junction or end point pixels.

The generalized correction of thinning defects consists of :

- (1) All doubly connected segments that are one pixel long are converted to junctions.
- (2) A perfectly closed thin line(loop) is broken into two segments by inserting two new junctions.
- (3) For any segment S_1 calculate the distance between its two ends(e_1, e_2) and call it d_1 , then for every pixel P_i of S_1 calculate its distance from the line connecting e_1 and e_2 , and call this distance d_2 . Now if $d_2 > d_1$ then split S_1 at P_i into segments S_2 and S_3 creating the new junction e_3 [19].

Figure 2.2.3-2 shows the thinned image after correction.

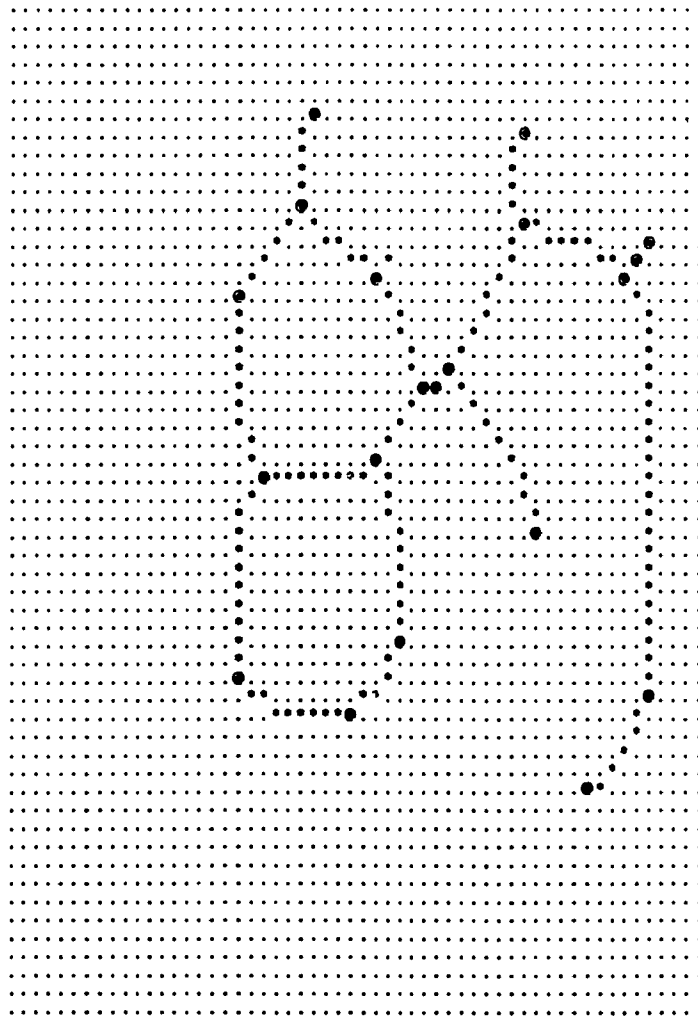


Figure 2.2.3-2: Marked thinned image after correction of thinning defects.

2.3- SKELETON LABELLING

After thinning, marking, and correction of thinning defects, the 0-valued pixels in $\text{Thinpix}(i,j)$ represent background and the 1-valued pixels are for foreground. Consequently, the segment pixels are all the remaining 1-valued pixels in $\text{Thinpix}(i,j)$ which are neither junction pixels nor end points, and the "program" can be of the following form:

```

For all pixels j and i Do
  JSpix(i,j) = Thinpix(i,j) ;copy
  If(Thinpix(i,j).Eq.1) Then
    compute n; (n is the no. of 1-valued pixels in N8)
    If (1≤n≤2) then JSpix(i,j)= -1
  Endif
Enddo i and j

```

which is abbreviated to:

$$\text{JSpix}(i,j) = \text{Junseg}(\text{Thinpix}(i,j))$$

In order to save computer memory space, the junctions will be given negative label numbers(i.e., -3, -4, ...) and the segments will be given positive label numbers(i.e., 4, 5, ...). We should notice that the selection of -3 and +4 as starting labels in our case is quite accidental and we could have started with any other negative and positive numbers. These numbers are used for the sake of representation and except the sign(negative or positive) the values do not carry any specific meaning. The connectivity in both cases is 8. Symbolically this labelling operation is indicated by:

$$\text{LJSpix}(i,j) = \text{LabelJS}(\text{JSpix}(i,j), \text{connectivity}=8)$$

where "labelJS" indicates the labelling operation and LJSpix(i,j) is the junction, end point and segment labelled image. However, note that the operation LabelJS is composed of two labelling operations, i.e.:

$$\text{LJ-pix}(i,j) = \text{Label}(\text{Thinpix}(i,j), \text{connect.}=8, \text{pixel-type}=1, \text{start-label}=4)$$

$$\text{LJSpix}(i,j) = \text{Label}(\text{LJ-pix}(i,j), \text{connect.}=8, \text{pixel-type}=-1, \text{start-label}=-3)$$

where LJ-pix(i,j) is the "junction labelled" image, see figure 2.3-1.

Now it is easy to determine the size of the junction regions and the lengths of the segments in the image. Actually, we can obtain two kinds of measures, namely, the size and length measured as a pixel count from LJSpix(i,j), and measured in terms of the average local thickness by using the values of the distance to the closest contour.

The processing of the skeleton information may be continued for determining segment shape, connectivity of segments across junctions to form lines, line shapes, etc., and also which segments should be changed or eliminated. The information regarding labels of segments, junctions, and end points are collected into a tabular structured form. Labels are used as relative addresses.

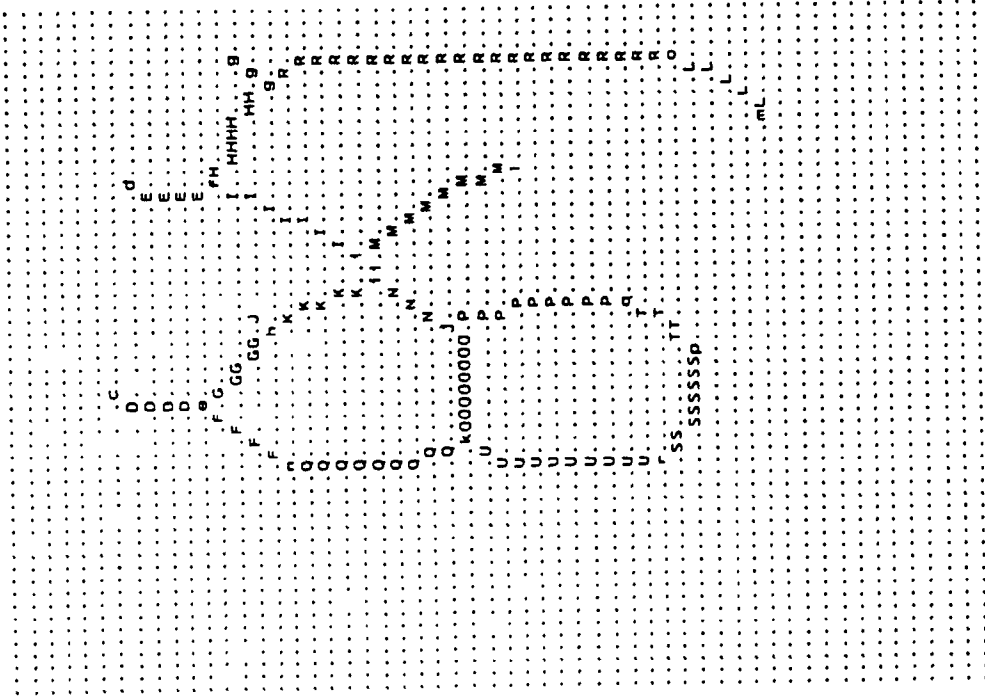


Figure 2.3-1(b):

Labelled skeleton image, LJSpix(i,j). The segment labels (4,5,6,...) are indicated by upper case letters (D,E,F,...). The junction labels (-3,-4,-5,...) are indicated by lower case letters (c,d,e,...)

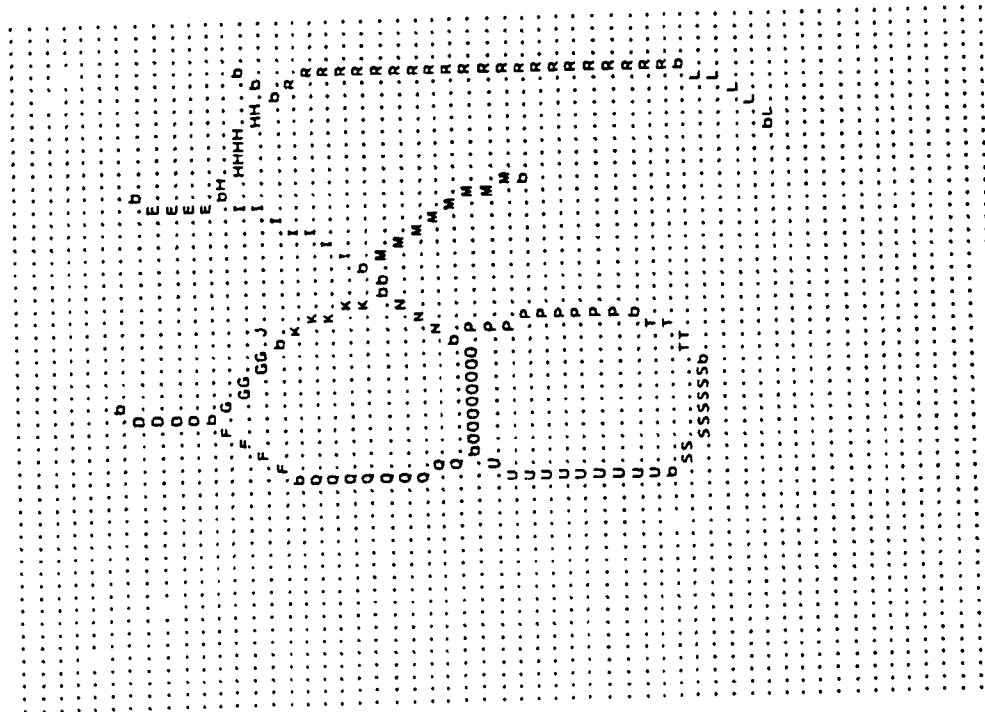


Figure 2.3-1(a):

Segment labelled image LJ-pix(i,j). The junctions are shown by b's and the segment labels (4,5,6,...) are indicated by upper case letters (D,E,F,...).

2.4- SEGMENTS LABELS DILATION

The purpose of dilation is to increase the size of the 1-valued or foreground regions in the image (or to increase the size of 0-valued background regions, but not both the zero and one valued regions at the same time). In each dilation cycle(iteration) one layer of external countour pixel is added to the 1-valued regions in the image. The sizes of the object are thus increased systematically by "sticking or plating" new edges onto the old edges layer by layer.

Constrained dilation is based on additional information derived from the local 3 by 3 neighbourhood N8 or N4, depending on the required connectivity, or the constraints are derived from other registered images. Additional control is obtained by using labelled images and forcing the operations to be carried out only on specific labels.

The unconstrained 8-connected dilation operator is based on the following logic within the 8-connected 3 by 3 neighbourhood N8, where P is the center pixel of N8 and $N8'$ is the periphery of N8. The input image is $Inpix(i,j)$ and the output image is $Outpix(i,j)$: " If the pixel P is on a 0-valued pixel in $Inpix(i,j)$, and there is at least one 1-valued pixel in $N8'$ of $Inpix(i,j)$, then $Outpix(i,j)=1$, else $Outpix(i,j)=Inpix(i,j)$."

The constraints that we impose onto the dilation operator is subject to a constraint image. In this case the dilation is only carried out if "allowed" by another registered image called $Mask(i,j)$. The mask image allows us to prevent the dilation from occurring in the regions where dilation is not wanted. Say dilation is allowed if the pixel in $Mask(i,j)$ is 1 and not allowed if the

pixel is 0.

Now the basic constrained 8-connected dilation operator has the following logic within the (8-connected) 3 by 3 neighbourhood $N8$, where P is the center pixel of $N8$ and $N8'$ is the periphery of $N8$. The input image is $LJSpix(i,j)$, the constraining image is $Inpix(i,j)$, and the output image is $Dilpix(i,j)$. The procedure consists of :

“If the pixel P is on a 0-valued pixel in $LJSpix(i,j)$, and the pixel P is on a 1-valued pixel in $Inpix(i,j)$, and there is at least one K -valued pixel in $(N8')$ of $LJSpix(i,j)$, and the setting of the pixel at P to K in $Outpix(i,j)$ does not change the connectivity among the differently labelled (non-zero) pixels in $N8$, then $Dilpix(i,j)=K$, else $Dilpix(i,j) =LJSpix(i,j)$.”

The results of segment labels dilation are shown in figure 2.4-1.

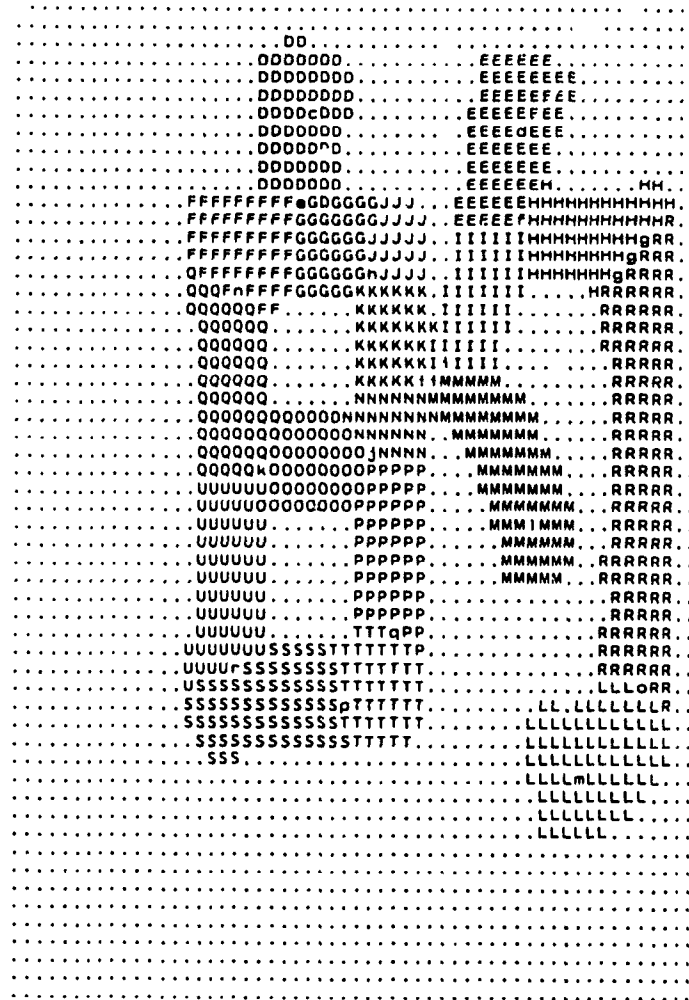


Figure 2.4-1: Dilatated segment labels image, Dilpix(i,j).

3.- FRAGMENTATION ALGORITHM

It is assumed that an object in the picture can be considered to consist of certain pieces, primitives or "atoms". A relatively simple object would consist of only one atom and a more complicated object of a certain arrangement of several atoms. The atom aggregates(molecules) may be further compounded into "macromolecules". At present each atom in the aggregate has the same weight.

Since the picture content is unknown, nor is there any knowledge available about the orientation, position, size, and distortion of the objects, a recognition algorithm can not be applied directly. However, it is possible to apply local differential operators to obtain gray level gradients(both in magnitude and direction), second derivative operators, local extremum operators for computing contour points, contour curvature operators, to compute local statistics on the above, etc. For the lack of a general name, the results of all such local operators will be called "point feature".

There is, of course, nothing novel in these computations. It should only be noticed that the point features can be computed without knowing what the picture contains and they are "attached" to the unknown objects and also to the background. When the objects in the picture are moved, the point features belonging to the objects "follow" the objects. Exceptions usually occur only at new contact boundaries between objects or when two previously touching or overlapping objects are separated. The problem really is to decide which of the possible computable point features are preferable and least subject to computational errors caused by noise. The selection of preferable point feature

requires also the knowledge of how the atoms are to be presented in the computer.

Since the number and type of the primitives or atoms out of which an object may be considered to be made up is rather indeterminate, they should not be left to direct human specification. It is desirable to invent an algorithm which, when operating on the point features, specifies both the location and the size of each atom. The algorithm is thus required to specify both the location of an atom and the neighbourhood in the picture which is to be called an atom, while the atom is still unknown (i.e. unrecognized).

Edge, line and corner detectors, template matching, etc. are applications of this idea when the algorithm is constructed to detect a particular atom. However, it is also possible to construct algorithms which, instead of detecting a particular shape, only detect neighbourhoods in the picture for which relatively coherent descriptions can be formulated independent of "what has happened" to the object in the picture. The medial axis transform is one such algorithm.

The primitives (atoms) are not chosen arbitrarily, nor are they chosen by operator. The computer is programmed to select its own atoms or primitives by applying an algorithm. The algorithm determines the location of the atom and what part of the object is to be considered as one atom. The following considerations[27] lead to this procedure:

- 1) Since no restriction have been placed on the number of objects in the picture, their location, size, rotation, distortion, etc., it is obvious that no template matching procedure will work in reasonable time. Contour following is also unlikely to succeed since the objects may touch or even overlap to a certain degree and may be "multiply connected".

- 2) Even though the present objects are black and white, this is no reason to make the procedures less general. Thus, the first question to be answered is: what useful information can be extracted from the picture if nothing is known about its contents?
- 3) Recognition of an unknown atom requires the information gathered about the atom, to be compared with some previously stored information. What form this comparison takes is not really too important. To make the comparison procedure practical, however, requires that the description of the atom is normalized and weighed in some sense before the comparison is carried out.
- 4) The number and type of atoms(primitives) out of which an object may be considered to be composed, is quite indeterminate, especially when the object becomes complex. Even for a triangle, the primitives may consist of a line(i.e. 3 lines arranged properly), a corner, two corners, or three corners, a corner with variable angle and the shape of the area inside the boundary. Which of descriptions is to be preferred?
- 5) The key to the problem of selecting the primitives is, whether or not they are recognized. This is not an impossibility. It does not really matter what these primitives actually are, even though it may be preferable to program the computer so that it selects primitives which agree with human opinions.

The above conditions are not easy to satisfy in a general case, and no general solution is claimed. The algorithm used in the present program does not use any of the mentioned point feature and works as following:

“Consider a junction point (x_j, y_j) in the picture as the origin of a polar co-ordinate system (r, θ) . In each angular direction θ ($\theta = 0^\circ, 6^\circ, 12^\circ, \dots, 354^\circ$) the radial distance r ($r_0, r_6, r_{12}, \dots, r_{354}$) to the pixel (contour pixel or skeleton pixel or both) which belongs to the segment connecting to this junction is measured. The (r, θ) description[21] of the neighbourhood, as seen from point x_j, y_j , thus resembles a radar picture of coast lines seen from a ship.”

All the necessary information for the algorithm are now available from the pre-processing steps. Labelled skeleton is in LJSpix(i,j);(see figure 2.3-1), labelled contours was obtained from Dilpix(i,j) and Distpix(i,j) at edges by "AND'ing" operation;(see figure 3-1) .

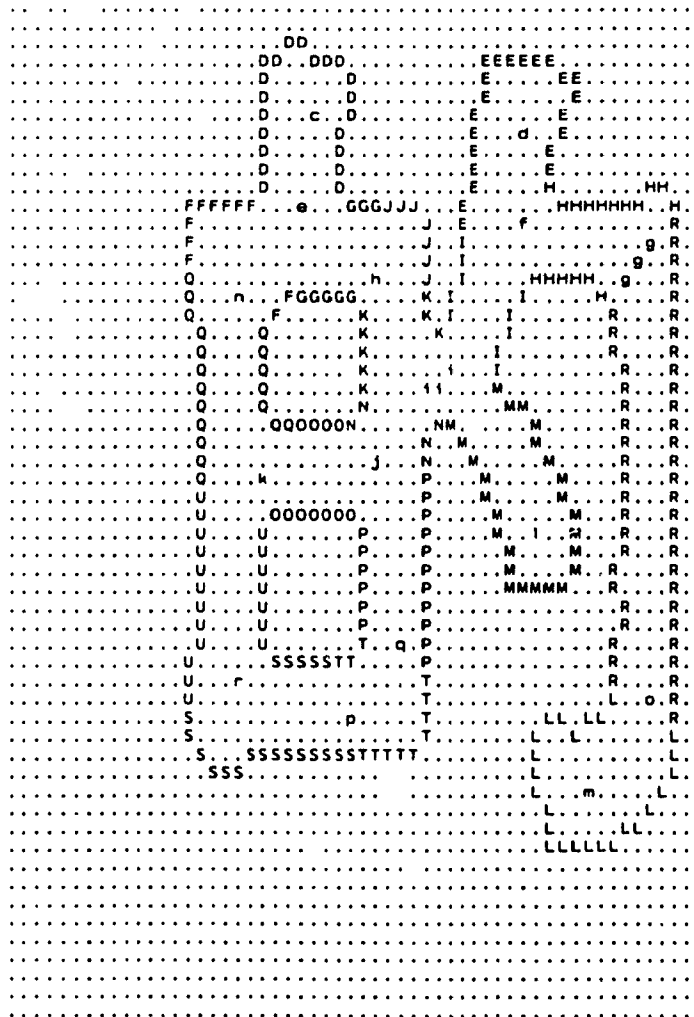


Figure 3-1: Labeled contour of the image.

3.1- (r,θ) DESCRIPTION

The algorithm defines neighbourhoods in the picture which will be called atoms after a proper description of the neighbourhood has been formed. The neighbourhood could be defined in terms of one of the following:

- 1- Contour pixels
- 2- Skeleton pixels
- 3- Both contour and skeleton pixels

Thus complicated object is automatically fragmented into atoms and computer has, so to say, selected its own atoms for the object. Obviously this procedure is meaningful only if the atoms so selected are the same for a given object, irrespective of where in the picture the object is located, how it is rotated and irrespective of what size it is. Variations in atoms will, of course, be caused by limited distortion of object shape, but this should not prevent the algorithm from finding the atoms.

To formulate a description for the atoms and to normalize the description before recognition is now a rather simple matter. By using a polar co-ordinate representation of the neighbourhood, the description consists of a set of 2-tuples (r,θ) around the junction points (x_j, y_j) . This is represented as a table:

x_j, y_j = Co-ordinates of junction point

θ, r = 2-tuples, one for each segment pixel (contour, skeleton, or both)

where θ = the angular direction, r = the radial distance to pixel

The procedure breaks down a complicated object into atoms, and normalizes the description of each atom. Theoretically speaking "ideal atoms" should exist, but since no attempt is made to "force" atom shapes onto the

machine, in practise it is only possible to show the machine images of "ideal atoms" from which, hopefully, the machine will extract the "ideal atoms". However, at present the problem of selecting ideal atoms is simply ignored since, it is not going to be a major programming problem to allow the machine to select its own ideal atoms.

The following discussion is based on the assumption that the machine's "memories" have been zeroes, i.e. it is completely "newborn". When the machine has located all atoms in the object, and normalized, weighed and matched them against each other, a supervised selection of "ideal atoms" starts.

There are two lists in the machine's memory. These are called "a list of presently seen atoms" , "a list of ideal or memory atoms" . The machine is programmed to proceed as follows:

- 1) The list of seen atoms is zeroed.
- 2) The atoms in the picture are found, normalized, and stored in the list of seen atoms.
- 3) The list of seen atoms are compared against the list of ideal atoms.
- 4) If there is an atom that does not compare well enough to any of the ideal atoms then insert this atom into the list of ideal atoms.

In order to handle segments variations, once the atom is extracted we chop it by a multiple of its average shortest distance to nearest contour pixels. Average shortest distance is computed by averaging the distances between the junction point and the nearest segment pixels, in this experiment the multiple value is set to 3, see figure 3.1-1. The extracted atoms are shown in figures 3.1-2, 3.1-3 and 3.1-4, where The atoms have been placed close to each other while still retaining the approximate shape of the original character. Since

atoms are built around junctions, the end points such as c,d,l and m have disappeared. Of course in the case of a lone segment the end points information(which are preserved and available) are of importance but since the concern of this project is with the junction points, so we leave the lone segments and end points out.

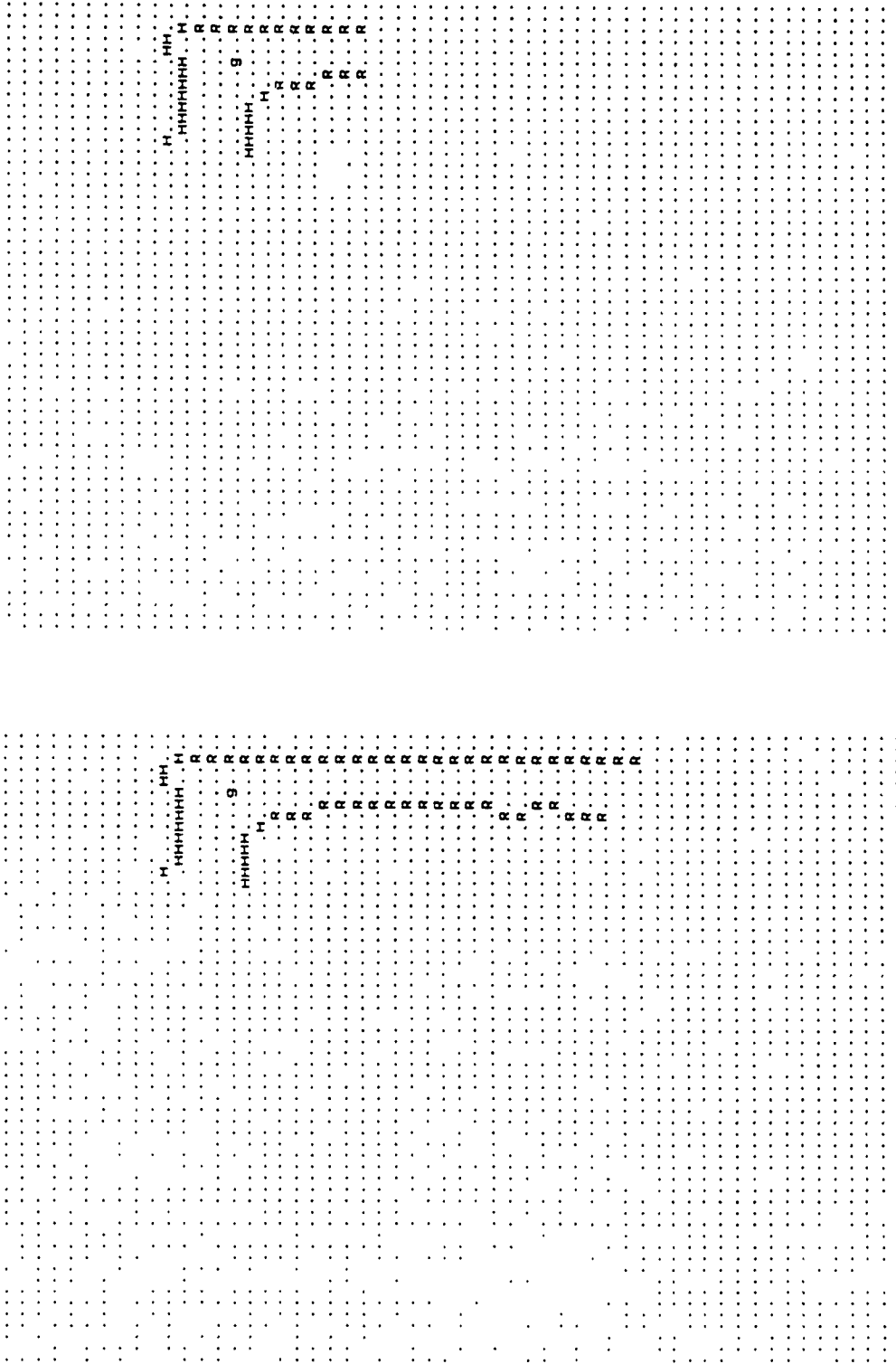


Figure 3.1-1: A "Contour" atom before and after chopping.

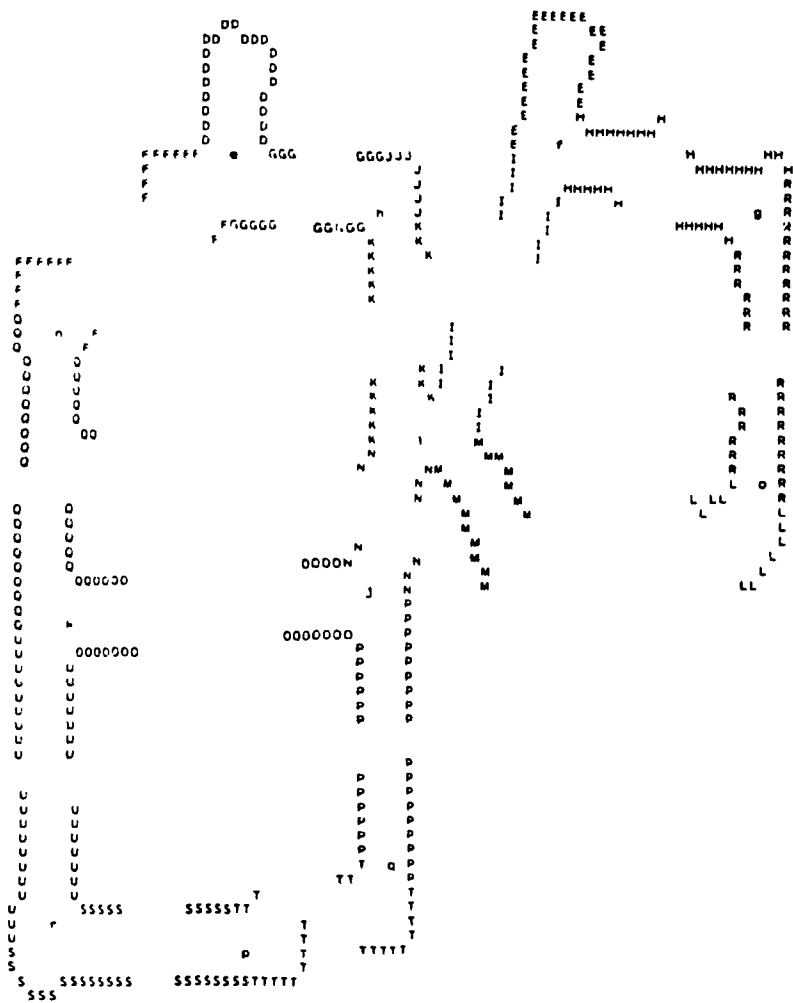


Figure 3.1-2: "Contour" atoms.

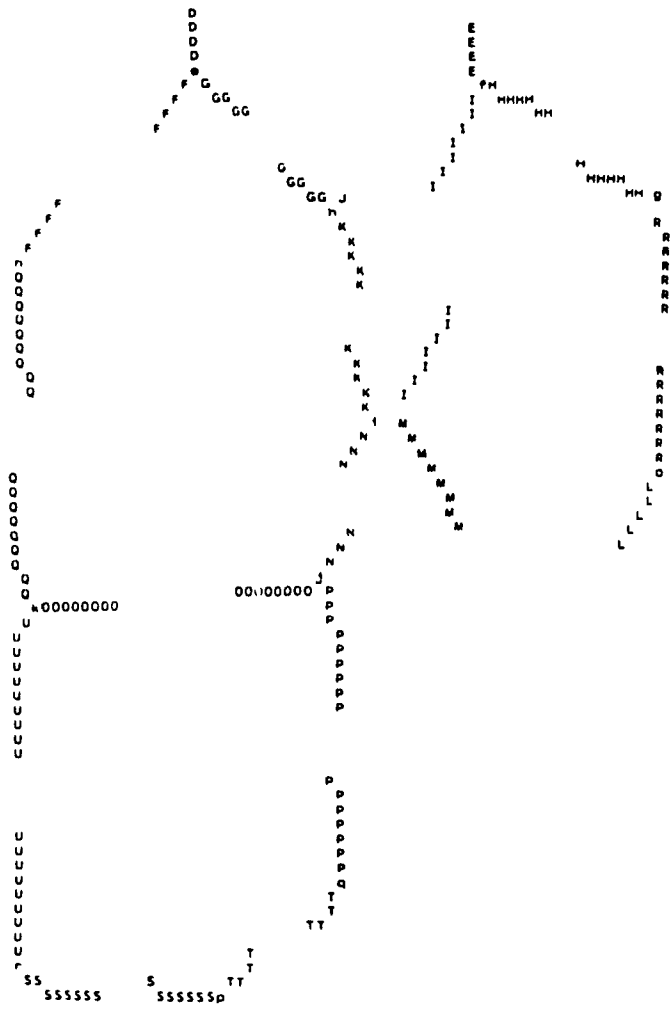


Figure 3.1-3: "Skeleton" atoms.

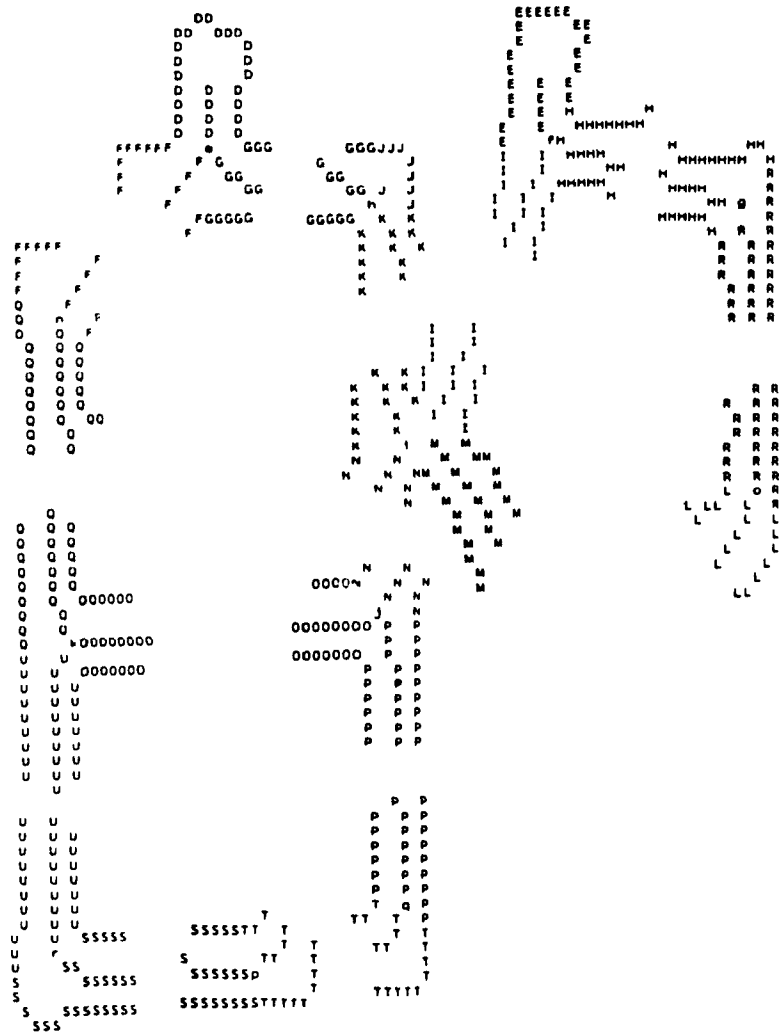


Figure 3.1-4: "Contour and Skeleton" atoms.

3.2- NORMALIZATION and WEIGHING

In many cases, patterns are normalized in the preprocessing process. Size, position, skew and line width are the types of normalization used. In fact size normalization is a very common technique used in pattern recognition[14]. Rotation(θ) can not be normalized but radii can be normalized. Since (r,θ) descriptions are multivalued functions(i.e. for the same value of θ we might have several radii) thus, we can not use correlation as a measure of similarity. To overcome this problem we have devised our own scheme which operates in two steps:

- a) Normalization and Weighing
- b) Matching

Where normalization and weighing are described in this section and matching will be described in the next section. The average radius is used to normalize the size, thus the (r,θ) description is transformed into a binary image of 37 by 60 pixels. Each (r,θ) description is transformed into a 1-valued entry (i,j) in the new image as following(see figure 3.2-1):

$$j = \theta/6^\circ$$

$$i = (r - r_{avr}) / ((r_{avr} - r_{std}) / nslots) + s$$

n = no. of 1-valued pixels

$$r_{avr} = \sum_{k=1}^n r_k$$

$$r_{std} = \text{SQRT} \left(\sum_{k=1}^n (r_k - r_{avr})^2 \right)$$

avr \equiv Average, std \equiv Standard deviation

nslots \equiv No. of slots = 6, s = 3.nslots + 1

By weighing we simply mean to assign a positive value to 0-valued pixels depending on how far they are from the 1-valued pixels, in other words this is a kind of distance measure. Weighing the new representation of (r, θ) description, is done in two phases. But before describing these two phases we need two definitions:

DEF.1- Zero-column (or 0-column) is a column which consists of zero's and nothing else.

DEF.2- One-column (or 1-column) is a column which has at least a one as one of its entries.

The two phases are as following:

- 1) Weigh zero-columns. The 0-columns are surrounded by two one-columns (j_1, j_2) , starting from column j_1 toward j_2 assign values (starting from 2,3,...) to all the entries of the 0-columns. Now do the reverse i.e., starting starting from j_2 toward j_1 assign values (starting from 2,3,...) to the 0-columns, providing that the new value is less than the previously assigned value.
- 2) Weigh one-columns. There are 3 cases
 - 2.1) If the 1-column has only One 1-valued entry in row i , then assign values (starting from 2,3,...) in both directions (upward & downward).

- 2.2) If the 1-column has two 1-valued entries(in rows i_1, i_2), then starting from i_1 toward i_2 assign values (starting from 2,3,...) to the pixels in the column. Now do the reverse i.e., starting from i_2 toward i_1 assign values to the pixels in the column providing that the new value is less than the previously assigned value. The entries in the column that fall beyond i_1 and i_2 are assigned values as it was mentioned in 2.1.
- 2.3) If the 1-column has more than two 1-valued entries then a combination of 2.1 and 2.2 is used.

Figure 3.2-2 shows the weighed atom.

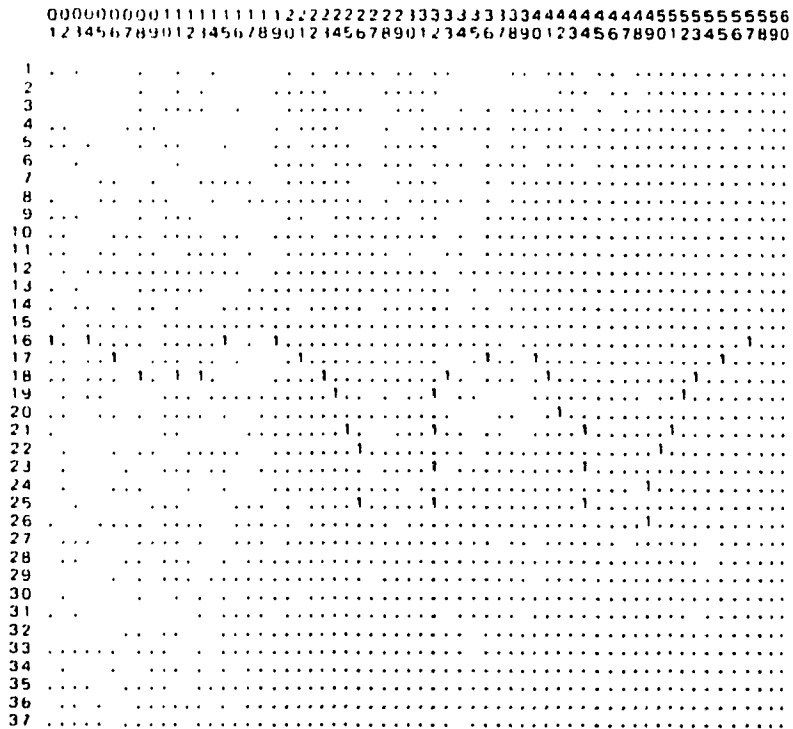


Figure 3.2-1: The normalized version of the "Contour" atom shown in fig. 3.1-1

```

0000000001111111122222222333333333444444445555555556
123456789012345678901234567890123456789012345678901234567890
1 6226272R2282826272627289122343298227232780212332421982726232
2 522526272272725232526278012343287226232679202332310872625232
3 422425262262624232425267902343276225232568292332209762524232
4 322324252252523232324256892343265224232457282332198652423232
5 2222324242422222223245782343254223323234627233208754232232
6 12212223223232123212223467234324322223223526233297643221232
7 02202122222220232021223562343232221232124252332865322120232
8 922920212212129232920212452343221220232013242332754212029232
9 822829202202028232829201342343210229232902232332643102928232
10 722728292292927232728290232343209228232891222332532092827232
11 622627282282826232627289122343298227232780212332421982726232
12 522526272272725232526278012343287226232679202332310872625232
13 422425262262624232425267902343276225232568292332209762524232
14 322324252252523232324256892343265224232457282332198652423232
15 22222124224242223222324578234325422323234627233208754232232
16 *22*2223223232*232*2223467234324322223223526233297643222*232
17 22222*2222222222222*22356234323222*232*24252332865322*22232
18 3223222*22*2*232323222*245234322*2222322*32423327542*2223232
19 4224232222222423242322*3423432*22232322232332643*22324232
20 52252423223232523252423223234322322423243*222332532232425232
21 622625242242426232625243*223432*42252325422*233242*342526232
22 7227262522525272327262542*23432252262326532223323*2452627232
23 8228272622626282328272653223432*62272327642*2332223562728232
24 922928272272729232928276422343227228232875222332*34672829232
25 0220292822828202320292875*23432*82292329862*2332245782920232
26 122120292292921232120298622343229220232097222332*56892021232
27 2222212022020222322212097323*230221232108232332267902122232
28 322322212212123232322210842343241222232219242332378012223232
29 42242322222224232423221952343252223232320252332489122324232
30 5225242322323252325242321062343263224232431262332590232425232
31 622625242242426232625243172343274225232542272332601342526232
32 722726252252527232726254282343285226232653282332712452627232
33 822827262262628232827265392343296227232764292332823562728232
34 922928272272729232928276402343207228232875202332934672829232
35 022029282282820232029287512343218229232986212332045782920232
36 1221202922929212321202986223432729270232097222332156892021232
37 222221202202022232221209732343230221232108232332267902122232

```

Figure 3.2-2: Weighed "Contour" atom , the 1-valued entries are shown by *'s and the numerals are mod(10).

3.3- MATCHING

Given that the normalization is consistent, irrespective of how the atom was originally located in the picture, the recognition of the atom is straight forward.

Thus, the conditions laid down in the beginning have all been met. The object can be fragmented algorithmically, atom descriptions can be obtained and these can be normalized and weighed before any recognition is attempted. The recognition of individual atom is now quite simple.

Each of the observable or non-ideal atoms a_u of a given type have an "idealized" version a^i from which they can be derived provided the size (S) and rotation (R) are known. In the analysis of a picture the situation is exactly opposite. It has been possible to construct a representation of an unknown but observable atom, a_u to a certain degree of accuracy, and it has been possible to normalize it for size, but not for rotation. However, it is known how to rotate the unknown atom a_u .

Assume that the machine "knows" a set of "ideal" or "memory" atoms $a_1^i, a_2^i, \dots, a_k^i, \dots, a_n^i$. The unknown atom a_u is normalized for size and weighed (a'_u) and compared with all the memory atoms. The comparison procedure consists of computing a weighted distance between a'_u and a_k^i , $k=1,2,\dots,n$, for all rotations θ , i.e.:

$$d_{uk}(\theta) = \text{distance}(a'_u, a_k^i); k=1,2,\dots,n; \theta=0 \text{ to } 2\pi$$

Distance between atoms a'_u and a_k^i if the rotation angle is θ_1 , is calculated as following:

Set tot_1, tot_2, n_1, n_2 to zero;
 For all pixels i_1 and j_1 Do
 If($a'_u(i_1, j_1).eq.1$)then
 Begin
 $n_1 = n_1 + 1$
 $tot_1 = tot_1 + a_k^i(i_1, j_1) - 1$
 Endif
 If($a_k^i(i_1, j_1).eq.1$)then
 Begin
 $n_2 = n_2 + 1$
 $tot_2 = tot_2 + a'_u(i_1, j_1) - 1$
 Endif
 Enddo i_1 and j_1

$$dis_1 = tot_1 / n_1$$

$$dis_2 = tot_2 / n_2$$

$$distance(a'_u, a_k^i) = (dis_1 + dis_2) / 2$$

If a'_u and a_k^i perfectly match then $distance(a'_u, a_k^i)$ would have a value equal zero, otherwise $distance(a'_u, a_k^i) > 0$ depending on how bad they mismatch ($distance(a'_u, a_k^i) \geq 0$).

The rotation angle (θ_k^*) which gives the minimum distance d_{uk}^* gives the best matching orientation between a'_u and a_k^i , i.e.:

$$d_{uk}^*(\theta_k^*) = \underset{(\theta)}{\text{minimum}} (d_{uk}(\theta)); k=1, 2, \dots, n$$

The comparison is carried out for all the ideal atoms in the memory. The unknown atom is considered to correspond to the ideal atom which has the lowest distance ($d^*(\theta^*)$), provided this distance is below a limit d_m i.e.:

$$d^*(\theta^*) = \underset{k}{\text{minimum}}(d_{uk}^*(\theta_k^*))$$

This procedure selects one ideal atom a^{i*} as being the one "nearest" to the unknown atom a'_u . If $d^*(\theta^*) \leq d_m$ then unknown atom a'_u is considered to be a^{i*} , but is rotated by angle θ^* and it has a certain size relation (S_u/S_i) with respect to a^{i*} (size of $a_u = S_u$, size of $a^{i*} = S_i$, size being defined as the average radius (\bar{r})). If $d^*(\theta^*) > d_m$ then the unknown atom is a new one and it is added to the machine's list of "ideal" atoms.

The results of matching for different type of atoms are shown in tables 3.3.1, 3.3.2 and 3.3.3. The values in each entry give the minimum distance for the best matching orientation. As pointed out before a zero shows a perfect match.

0	0.000	0.853	1.000	1.308	1.137	1.149	1.006	0.921	0.971	0.954	1.073	0.890
0	0.853	0.000	0.817	1.227	1.195	1.053	0.811	1.244	1.059	1.194	1.100	0.794
0	1.000	0.817	0.000	1.281	1.235	1.209	0.918	1.134	1.069	1.201	1.140	0.635
0	1.308	1.227	1.281	0.000	1.182	1.375	1.286	1.129	1.291	1.225	1.247	1.272
0	1.137	1.195	1.235	1.182	0.000	1.119	1.080	1.159	1.301	1.347	1.231	1.264
0	1.149	1.053	1.209	1.375	1.119	0.000	1.236	1.203	1.165	1.427	1.430	0.925
0	1.006	0.811	0.918	1.286	1.080	1.236	0.000	1.269	1.128	1.251	1.275	0.868
0	0.921	1.244	1.134	1.129	1.159	1.203	1.269	0.000	1.057	1.241	1.147	1.042
0	0.971	1.059	1.069	1.291	1.301	1.165	1.128	1.057	0.000	1.150	0.481	1.157
0	0.954	1.194	1.201	1.225	1.347	1.427	1.251	1.241	1.150	0.000	0.959	1.091
0	1.073	1.100	1.140	1.247	1.231	1.430	1.275	1.147	0.481	0.959	0.000	1.194
0	0.890	0.794	0.635	1.272	1.264	0.925	0.868	1.042	1.157	1.091	1.194	0.000

TABLE 3.3-1: Matching results of "Contour" atoms.

0.000	1.376	3.478	1.647	2.801	1.184	1.341	2.679	3.353	2.531	2.969	3.349
1.376	0.000	2.778	1.480	1.977	1.417	1.253	3.167	3.583	3.134	3.081	2.556
3.478	2.778	0.000	3.712	3.317	2.292	3.841	1.625	2.000	1.780	1.856	1.177
1.647	1.480	3.712	0.000	2.488	1.394	1.545	2.311	1.864	2.045	1.864	3.517
2.801	1.977	3.317	2.488	0.000	1.969	1.950	3.384	3.473	3.425	3.310	2.886
1.184	1.417	2.292	1.394	1.969	0.000	1.439	3.208	3.236	3.280	3.197	2.823
1.341	1.253	3.841	1.545	1.950	1.439	0.000	3.811	3.746	3.295	3.682	3.321
2.679	3.167	1.625	2.311	3.384	3.208	3.811	0.000	0.708	0.742	0.352	1.177
3.353	3.583	2.000	1.864	3.473	3.236	3.746	0.708	0.000	1.424	0.439	2.125
2.531	3.134	1.780	2.045	3.425	3.280	3.295	0.742	1.424	0.000	1.045	1.284
2.969	3.081	1.856	1.864	3.310	3.197	3.682	0.352	0.439	1.045	0.000	1.452
3.349	2.556	1.177	3.517	2.886	2.823	3.321	1.177	2.125	1.284	1.452	0.000

TABLE 3.3-2: Matching results of "Skeleton" atoms.

	0.000	1.133	1.143	1.255	1.116	1.183	0.881	0.749	1.088	1.035	1.122	0.960
	1.133	0.000	1.070	1.252	1.138	1.169	1.145	1.084	1.217	1.178	1.164	1.092
	1.143	1.070	0.000	1.036	0.982	1.138	0.905	1.060	1.111	1.179	1.144	0.832
	1.255	1.252	1.036	0.000	1.215	1.273	1.150	1.143	1.180	1.154	1.167	1.077
	1.116	1.138	0.982	1.215	0.000	1.095	1.016	1.078	1.053	1.206	1.125	1.094
	1.183	1.169	1.138	1.273	1.095	0.000	1.149	1.186	1.196	1.292	1.296	1.174
	0.881	1.145	0.905	1.150	1.016	1.149	0.000	1.006	1.047	1.129	1.105	0.864
	0.749	1.084	1.060	1.143	1.078	1.186	1.006	0.000	1.038	0.924	1.079	1.014
	1.088	1.217	1.111	1.180	1.053	1.196	1.047	1.038	0.000	1.052	0.369	0.951
	1.035	1.178	1.179	1.154	1.206	1.292	1.129	0.924	1.052	0.000	1.014	1.150
	1.122	1.164	1.144	1.167	1.125	1.296	1.105	1.079	0.369	1.014	0.000	0.950
	0.960	1.092	0.832	1.077	1.094	1.174	0.864	1.014	0.951	1.150	0.950	0.000

TABLE 3.3-3: Matching results of "Contour and Skeleton" atoms.

4. ATOMS INTERRELATIONSHIP AND OBJECT RECOGNITION

After pattern primitives are selected, the next step is the construction of a grammar(or grammars) which will generate a language(or languages) to describe the patterns under study. It is known that increased descriptive power of a language is paid for in terms of increased complexity of the syntax analysis system(recognizer or acceptor). Finite-state automata are capable of recognizing finite-state languages, although the descriptive power of finite-state language is also known to be weaker than that of context-free and context-sensitive languages. On the other hand, nonfinite, nondeterministic procedures are required, in general, to recognize languages generated by context-free and context-sensitive grammars. The selection of a particular grammar for pattern description is affected by the primitives selected, and by the tradeoff between the grammar's descriptive power and analysis efficiency.

If the primitives selected are very simple, more complex grammars may have to be used for pattern description. On the other hand, the use of sophisticated primitives may result in rather simple grammars for pattern description, which in turn will result in fast recognition algorithms. The interplay between the complexities of primitives and of pattern grammars is certainly very important in the design of a syntactic pattern recognition system. Context-free programmed grammars, which maintain the simplicity of context-free grammars but can generate context-sensitive languages, have recently been suggested for patter description[6].

A number of special languages have been proposed for the description of patterns such as English and Chinese characters, chromosome images, spark chamber pictures, two-dimensional mathematics, chemical structures, spoken words and finger print patterns[4,22]. For the purpose of effectively describing high dimensional patterns, high dimensional grammars such as Web grammars, array grammars, graph grammars, tree grammars, and shape grammars have been used for syntactic pattern recognition[4,23,24].

Ideally speaking, it would be nice to have a grammatical(or structural) inference machine which would infer a grammar from a given set of patterns. Unfortunately, not many convenient grammatical inference algorithms are presently available for this purpose. Nevertheless, recent literatures have indicated that some simple grammatical inference algorithms have already been applied to syntactic pattern recognition, particularly through man-machine interaction[25,27].

To describe the relationship between the various atoms that constitute an object is now relatively straightforward. The description, however, has to be "relativistic". The situation is best illustrated with an example. Assume that an object consists of ideal atoms a_1^i and a_2^i . In the object a_1^i occurs in two different sizes and rotations, i.e., a_1^i has two present(observable) realizations, i.e., $a_1^1 = a_1^i(S_1, \lambda_1)$, where S_1 =size w.r.t. a_1^i ; λ_1 =rotation w.r.t. a_1^i . If a_1^1 is to be used as a starting point for locating another atom say a_1^2 , then the distance $d_{1,2}$ and the angular direction $\psi_{1,2}$ where to find a_1^2 , have to be given with respect to a_1^1 . If the object is magnified by a factor m , the size(S_1) of a_1^1 is m times larger and the distance $d_{1,2}$ is also m times larger. Thus a description of where the next atom is to be located can be given in

terms of $d_{1,2}/S_1$ and $\psi_{1,2}$, measured with respect to λ_1 . The machine is now able to estimate the location of the next atom irrespective of the size and rotation of the object. The distortion of the object or inaccurate location of a_1^1 only introduces an error term into the estimation where a_1^2 is to be found[28].

In the recognition part, the taught relationships are used to guide the machine from the identified atoms to the expected atoms until an object of the desired class has been identified or all the usable taught relationships have been exhausted. Due to the limits of this work, the atoms interrelationship and object recognition parts have not been implemented.

5.- RESULTS OF EXPERIMENTS

Two experiments were implemented which will be described in the following:

EXPERIMENT I :

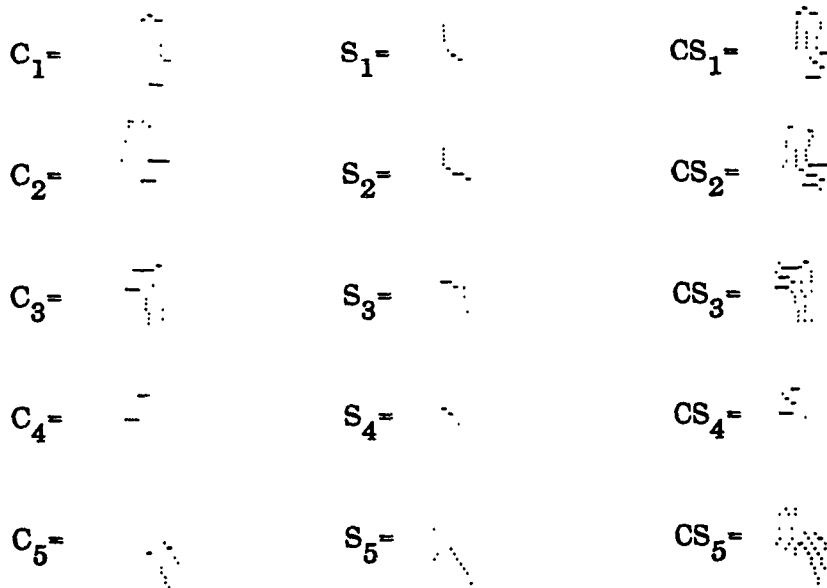
In this experiment we will verify which of the representation scheme is most reliable and best representative of the atom. Referring to tables 3.3-1, 3.3-2 and 3.3-3, if we set the matching threshold to 0.25 then all three schemes are equivalent and each will classify 12 unique atoms which will be referred to by using the following conventions:

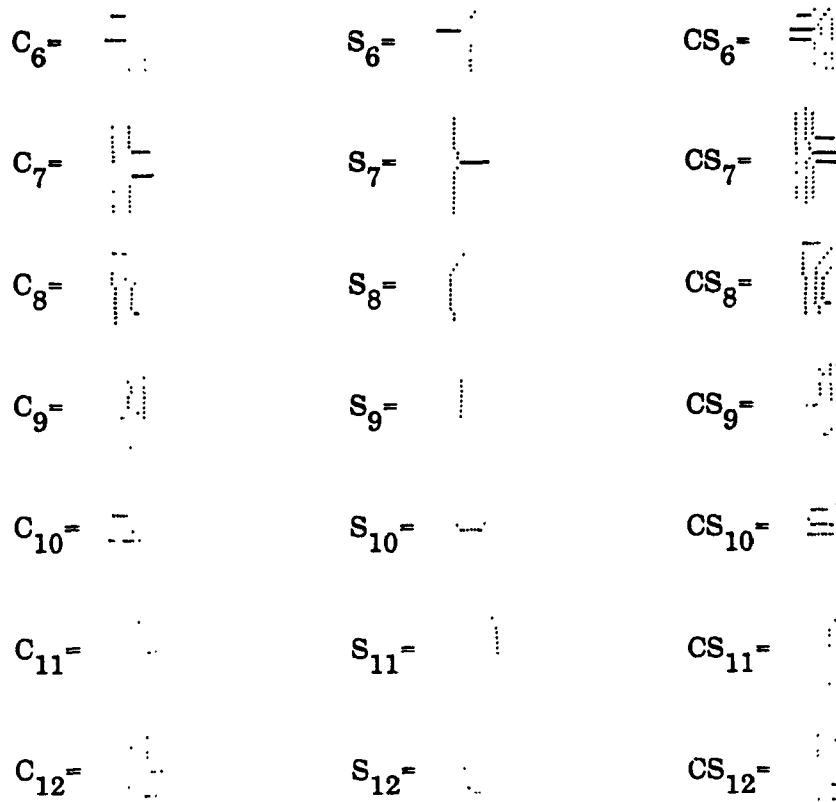
C_i = "Contour" atom no. i

S_i = "Skeleton" atom no. i

CS_i = "Contour and Skeleton" atom no. i

The atoms are as following:





Starting with "Contour" representation we will increase the threshold and examine how each representation classifies atoms. At threshold=0.50 atoms C_9, C_{11} are in the same class which is a meaningful classification, at threshold set to 0.75, atoms C_3 and C_{12} are classified similar which is a reasonable classification. If we increase the threshold to 1.00 we will see that atoms $C_1, C_2, C_3, C_6, C_7, C_8, C_{10}, C_{12}$ are signaled similar implying that atoms of 2-segments and 3-segments are similar, which is not an acceptable classification at all. By increasing the threshold to 1.25, 1.50, ..., we will see that the classification degrades even further by putting the 2-segment, 3-segment and 4-segments in the same class, so we conclude that "Contour" representation is not a good representation.

Next we verify "Contour and Skeleton" atoms. At threshold set to 0.50 atoms CS_9 and CS_{11} are classified similar. At threshold set to 0.75 atoms CS_1 and CS_8 are classified similar implying that a 3-segment atom is similar to a 2-segment atom. This representation even degrades more at threshold set to 1.00 by classifying atoms $CS_3, CS_5, CS_7, CS_{12}$ similar, implying that 2-segment, 3-segment and 4-segment atoms are similar. The same trend would follow by increasing the threshold to 1.25, 1.50, ..., implying that this is not a good representation scheme.

Finally we come to "Skeleton" atoms, at threshold set to 0.50 atoms S_8, S_9 and S_{11} are classified similar which is quite a good classification. At thresholds 0.75, 1.00 nothing will change but at threshold set to 1.25 we will see new classes, first, atoms S_1 and S_6 , second, atoms S_2 and S_7 , and third, atoms S_3 and S_{12} are signaled similar which is a very good classification. By increasing the threshold to 1.50, 1.75, ..., the classification starts degrading, but we observed that at threshold set to 1.25 the classification of atoms to similar classes was acceptable and good so we conclude that the most reliable and best representative of the property of atoms is the "Skeleton" representation.

EXPERIMENT II :

We observed in experiment I that "Skeletons" are most reliable in representing atoms. In experiment II we passed all the 50 characters in the data set (see appendix I) through the fragmentation program and based on the discussion in section 3.3 about ideal and non-ideal atoms and by using a supervised selection of ideal atoms constructed a data base of ideal atoms. We

observed that a set of eleven atoms (see figure 5-1) were sufficient to identify and recognize all the atoms produced by the fragmentation procedure. Since the characters were too small, fat, and sometimes had touching parts, some of the atoms look similar or consist of a few pixels. We believe that an automated selection of ideal atoms will produce more accurate results.

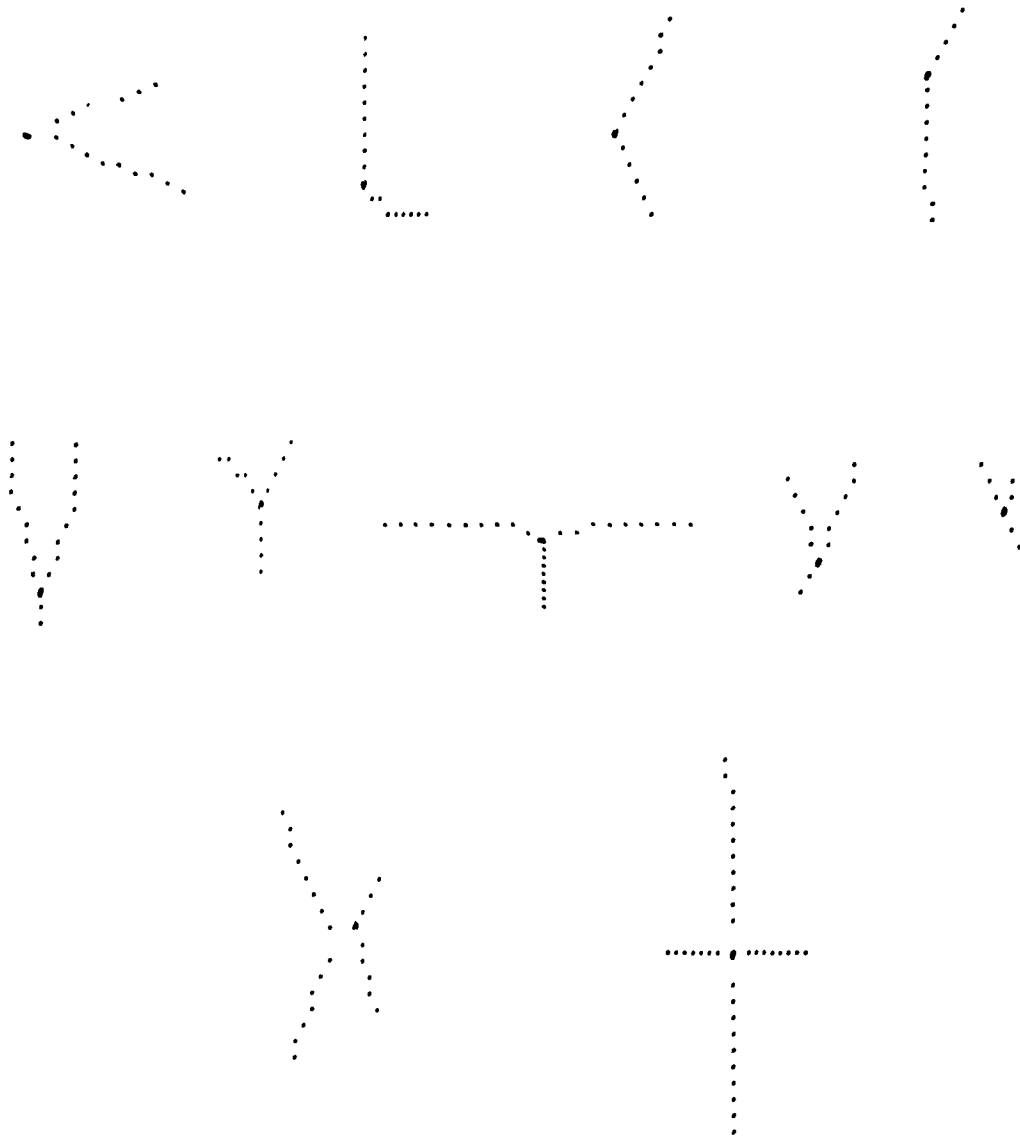


Figure 5-1: Ideal atoms.

6.- CONCLUSION

We have defined a method for fragmenting unconstrained line structures (i.e., Chinese characters) which is very simple and straightforward. After preprocessing, the junction points are selected as the center points of primitives. Each primitive(atom) is constructed out of a junction and the segments connecting to it. We also implemented two experiments.

In the first experiment we verified which of the representation scheme(skeleton, contour, or both) is most reliable and best representative of the atom. We observed that the skeleton of such structures are more reliable and carry more information than the contour or a combination of contour and skeleton.

In the second experiment we passed 50 Chinese character through the process to see how many different atoms are there. We observed that almost a handful of such atoms is sufficient to define any structure .

Our method is far from complete and we believe that there is a lot of room for improvement in any aspects. The characters(data set) used for our work were too small, fat and sometimes had touching parts which caused problems. Larger and thinner characters, and a better thinning algorithm which preserves the main characteristics of the line structure would improve the results drastically.

7.- REFERENCES

1. AHO A. V. and PETERSON T. G., "A Minimum Distance Error-Correcting Parser for Context-Free Languages", SIAM journal on computing, Vol. 4, Dec. 1972.
2. AHO A. V. and ULLMAN J. D., The Theory of Parsing, Translation, and Compiling, Vol. 1, Prentice-Hall, 1972.
3. PAVLIDIS T., Structural Pattern Recognition, Springer-Verlog, 1977.
4. FU K. S., Syntactic Pattern Recognition and Applications, Prentice Hall, 1982.
5. CHEN C. H., "On Statistical and Structural Feature Extraction" in Pattern Recognition and Artificial Intelligence, ed. by C. H. CHEN, Academic Press, 1976.
6. FU K. S., Syntactic Methods in Pattern Recognitions, Academic Press, 1974.
7. FU K. S., Digital Pattern Recognition, Springer Verlog, 1980.
8. GRIMSDALE R. L., SUMMER F. H., TUNIS C. J., KILBURN T., PROC. IEE 106b, 210(1959); represented in Pattern Recognition, Wiley, Newyork, PP. 317-338, 1966.
9. EDEN M., HALLE M., PROC. 4th London SYMP. on Information Theory, Butterworth, London, PP. 287-299, 1961.
10. STALLINGS W., Computer Graphics and Image Processing 1, PP. 47, 1972.
11. SAKAI T., GAO M., TERAJ H., Information Processing 10, PP. 10, 1970.
12. CHANG K. S., IEEE Trans. Systems, Man and Cybernetics. Vol. 3,

- PP. 257, 1973.
13. LIU Y., KASVAND T., "A New Approach to Machine Recognition of Chinese Characters", IEEE 7'th International Conference on Pattern Recognition Proceedings, PP. 381-384, 1984.
 14. SUEN C. Y., "Distinctive Features in Automatic Recognition of Handprinted Characters", Signal Processing, Vol. 4, PP. 193-207, April 1982.
 15. SHAPIRO L. G., HARALICK R. M., "Decomposition of Two-Dimensional Shapes by Graph Theoretic-Clustering", IEEE Trans. on Pattern Recognition and Machine Intelligence, Vol. PAMI-1, No. 1, PP. 10-19, 1979.
 16. GUERRA C., PIERONI G. G., "A Graph Theoretic Method for Decomposing Two-Dimensional Polygonal Shapes into Meaningful Parts", IEEE Trans. on Pattern Recognition and Machine Intelligence, Vol PAMI-4, NO. 4, PP. 405-407, 1982.
 17. PAVLIDIS T., "Analysis of Set Patterns", Pattern Recognition, Vol. 1, PP. 165-178, 1968
 18. BJORKLUND C., PAVLIDIS T., "Global Shape Analysis by K-syntactic Similarity", IEEE Trans. on Pattern Recognition and Machine Intelligence, Vol. PAMI-3, NO. 2, PP. 144-154, 1981 .
 19. PAVLIDIS T., "Structural Pattern Recognition Application", Springer Verlag, PP. 11-45, 1977.
 20. HILDITCH C. J., "Linear Skeleton from Square Cupboards", Machine Intelligence IV, eds. B. MELTZER and D. MICHIE, PP. 403-420, Edinburgh University Press, Edinburgh, 1969.

21. KASVAND T., "Computer Simulation of Pattern Recognition by Using Cell-Assemblies and Contour Projection", Proc. of IFAC TOKYO symposium, PP. 203-210, 1965.
22. FU K. S., Syntactic Pattern Recognition Applications, Springer-Verlog, 1977
23. FU K. S., "Tree Languages and Syntactic Pattern Recognitions", in Pattern Recognition and Artificial Intelligence, ed. by C. H. CHEN, Academic Press, 1976.
24. ROSENFELD A., Picture Languages, Academic Press, 1979.
25. LU S. Y. and FU K. S., "Stochastic Tree Grammars Inference for Texture Synthesis and Discrimination", Computer Graphics and Image Processing, Vol. 9, March 1979.
26. YOU K. C. and FU K. S., "A Syntactic Approach to Shape Recognition Using Attributed Grammars", IEEE Trans. on Systems, Man and Cybernetics, Vol. 9, June 1979.
27. KASVAND T., "Some Thoughts on Picture Languages" in Methodologies of Pattern Recognition, ed. by S. WATANABE, Academic Press, PP. 333, 1969.
28. KASVAND T., "Experiments with an On-Line Picture Language", in Frontier of Pattern Recognition, ed. by S. WATANABE, 1971.
29. KASVAND T., "Segmentation Using Thin Lines", Digital Signal Processing, PP. 882-889, 1984.

8.- APPENDIX I

The data set consists of the following 50 Chinese characters.



