AN ALGORITHM FOR SEPARATING UNIMODAL FUZZY SETS ON A GRID AND ITS APPLICATION TO OBJECT ISOLATION AND CLUSTERING

Thesis for the Degree of M. S.
MICHIGAN STATE UNIVERSITY
ROBERT LEWIS WALTON
1973

ABSTRACT

AN ALGORITHM FOR SEPARATING UNIMODAL FUZZY SETS
ON A GRID AND ITS APPLICATION
TO OBJECT ISOLATION AND CLUSTERING

By

Robert Lewis Walton

The goal of object isolation is to separate a scene into "regions of interest", each of which corresponds to an object or a portion of an object. The objective of this thesis is to develop a machine algorithm capable of isolating objects in an image plane. Such an algorithm can also be applied to the problem of clustering points in a feature space by treating a "density function" of the points as the intensity function of a scene and then finding the "objects" present. These objects correspond to clusters.

A basic approach to the object isolation problem was presented in terms of a clustering algorithm developed by Gitman and Levine[1]. Their algorithm is capable of separating a fuzzy set into unimodal regions and is capable of performing object isolation, provided each object can be made to correspond to a unimodal fuzzy set. Usually, if the input scene is preprocessed by low-pass filtering, each object can be made to correspond to one (or perhaps several) unimodal sets. Normally, at least several hundred data points are required to represent

the objects in a scene. If the assumption of a constant number of points per symmetric subset [1] is made, the computational requirements increase at least as rapidly as n^2, where n is the number of data points. For a typical number of data points, the computational requirements are too large to base a practical object isolation scheme on this algorithm. In addition, the algorithm has deficiencies related to equally spaced points and points of equal magnitude.

Several attempts are presented in this thesis for overcoming these deficiencies. As a result of these attempts, a new algorithm, the Unimodal Tree Algorithm, was developed. This algorithm assumes that the data points lie on a uniform grid, which is usually the case when a scene is scanned. It has computational and storage requirements which increase linearly with the number of grid points. The regions the algorithm generates are proven to be unimodal, and the union of any two regions is shown to be non-unimodal. Furthermore, these unimodal regions may be of any shape. There is no dependence upon spherical or elliptical regions.

Several experiments involving the Unimodal Tree Algorithm were performed. A scene containing three spheres and three ellipsoids was scanned and processed using the algorithm. The result was six regions, each of which contained one object. Another scene which was processed contained two touching ellipsoids. These objects were also separated. A third scene containing two mites in contact was run; three regions resulted. (One mite was light at both ends and darker in the center.)

Several bivariate Gaussian point sets were run using the Unimodal Tree Algorithm as a clustering scheme (in conjunction with a routine to form a density function on a grid). Even severely overlapping Gaussian clusters were separated. A crescent-shaped cluster with a Gaussian cluster in its center was run to demonstrate the independence of the algorithm with respect to cluster shape. These examples were all run on a "mini-computer" (IBM 1800). Separation of a 25 x 25 scene into unimodal regions required about one minute of IBM 1800 time. A CDC 6500 (large computer) separated these same scenes in approximately 1.5 CPU seconds per scene.

---

[1] Israel Gitman and Martin D. Levine, "An Algorithm for Detecting Unimodal Fuzzy Sets and Its Application as a Clustering Technique," IEEE Transactions on Computers, Vol. C-19, No. 7, pp. 583-593, July 1970.

AN ALGORITHM FOR SEPARATING UNIMODAL FUZZY SETS ON A GRID AND ITS APPLICATION TO OBJECT ISOLATION AND CLUSTERING

By

Robert Lewis Walton

A THESIS

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Department of Electrical Engineering and Systems Science

1973

ACKNOWLEDGEMENTS

I thank Dr. P. David Fisher for his invaluable help and suggestions during the writing of this thesis. The large amount of time he devoted to my instruction is greatly appreciated. The help of Dr. R. C. Dubes and Dr. John Kreer in proof reading the document is also greatly appreciated.

I also thank my working wife, Marsha, for her help and support during the preparation of this thesis. Her understanding and sacrifices have been of great help.

Finally, I thank Linda Swan for doing a fine job of typing the manuscript.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

CHAPTER I

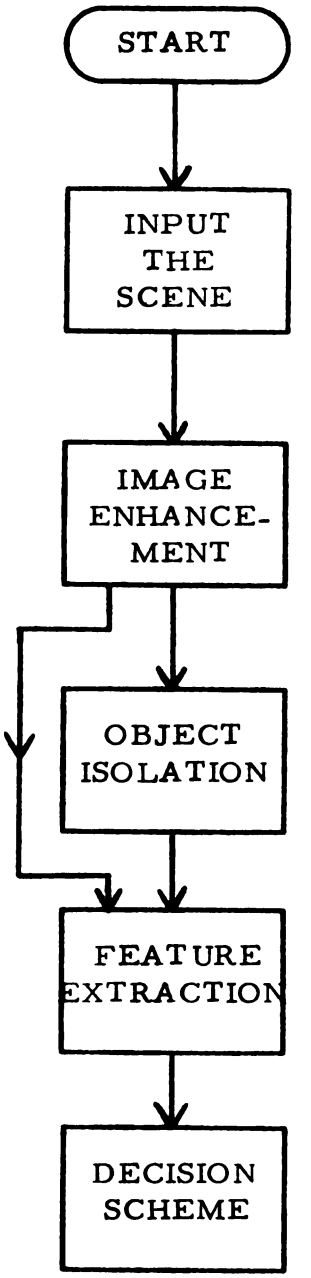
INTRODUCTION

1.1 Object Isolation Problem

The process of classifying objects in an image using a machine may be divided into four steps (Nagy [8]): image enhancement, object isolation, feature extraction, and, finally, application of a decision scheme. Image enhancement is normally necessary as a pre-processing step to put the raw input data into a suitable form for the steps which follow. Object isolation separates the image into regions containing one object or a portion of an object. This step will allow the "background" to be discarded, thus potentially saving much computer storage and computation later in the object classification procedure. The feature extraction procedure extracts information useful in classifying the objects, and the decision scheme determines the classification of each object. This object classification procedure is illustrated in Figure 1.

Separating objects in a picture is a relatively simple procedure for the human mind. An example of such a problem is the separation of ellipsoids resting on a dark flat surface, as shown in Figure 2. A 'simple'machine algorithm to separate ellipsoids might not work properly if the ellipsoids are of different size and orientations, or if some of them are touching. Complex algorithms to accomplish object

Figure 1: Object Isolation in an Object Classification Scheme



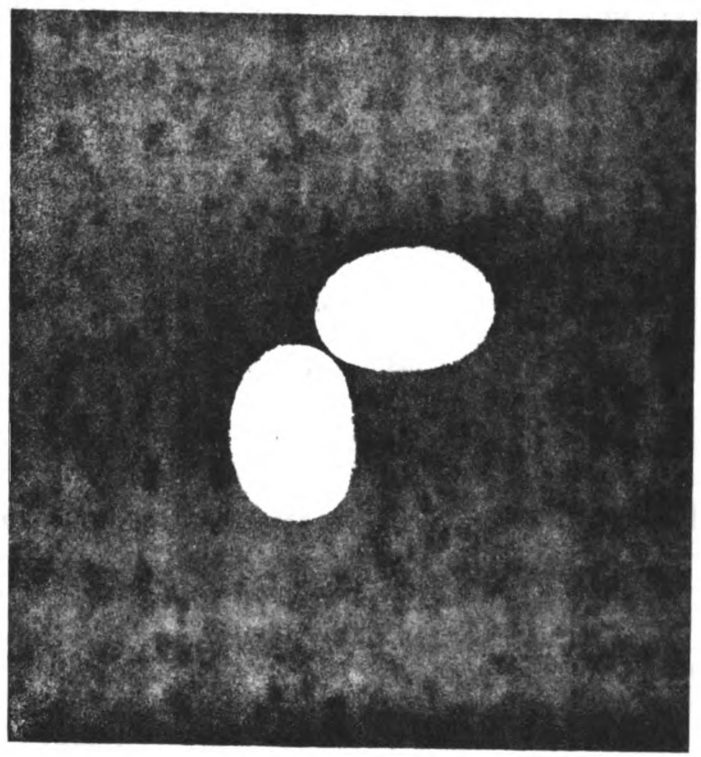


Figure 2. Two Touching Ellipsoids.

isolation under such adverse conditions may be very costly to operate in terms of computational requirements and/or storage requirements.

1.2 Machine Algorithms

A machine method applicable to object isolation has been suggested by Gitman and Levine [5]. The method they discuss is capable of separating touching ellipsoids, and is virtually unaffected by the size, shape, or orientation of the ellipsoids. Their algorithm separates an image into regions having only one "hill" in the intensity function (brightness level) of the image. Such a partitioning yields the separation required for object isolation.

Separating a scene into "unimodal" regions is useful in object isolation, but Gitman and Levine's algorithm has some deficiencies. The first deficiency is that special processing is required for equally-spaced sample points. When images are sampled on a grid of some sort, all the data requires special processing in the algorithm. The second deficiency is the special handling required by points of equal intensity-function value. If many such points occur, a large increase in computation time will result. Equal-intensity points will occur many times in an image having only a few brightness quantization levels.

1.3 Objectives and Accomplishments of the Thesis

The primary objective of this thesis is to present an algorithm capable of partitioning a rectangular grid into regions which are unimodal with respect to a function defined at each grid point, and to demonstrate its utility by performing several object isolation experiments

using the algorithm. Several properties which are desirable for an object isolation algorithm to possess are the following:

1. The algorithm should be capable of detecting objects of general shape.
2. Objects should not be skipped or missed.
3. Each object should generate one region, with no spurious "extra" regions due to quantization of the intensity function.
4. Computation and storage requirements should increase at most linearly with the number of grid points since a large number of points (500 or more) are usually needed to represent the objects.

The algorithm suggested by Gitman and Levine [5] is capable of generating the type of separation necessary for object isolation, but objects may be missed by the algorithm if they have a "flat peak" in intensity value. Also, the computation time required by Gitman and Levine's algorithm increases at least as fast as n^2, where n is the number of data points (assuming a constant number of points per symmetric subset). The amount of storage required for Gitman and Levine's algorithm is not fixed with respect to the number of data points due to a number of variable-length lists which are formed. Storage requirements for these lists may be several times the number of grid points. Several modifications of this algorithm and some new ideas were introduced in an effort to obtain an algorithm possessing the above properties.

The Subset Uniting Procedure was developed to eliminate the problem of skipping objects with a "flat peak". This procedure replaces the

last two steps of Gitman and Levine's algorithm, retaining only the first step. This procedure did eliminate the skipping of objects, but the computational requirements were still increasing rapidly with the number of points. Gitman and Levine's algorithm and the Subset Uniting Procedure are discussed in Chapter II.

Some simplifications of Gitman and Levine's algorithm as well as the Subset Uniting Procedure are possible when working on a grid, as is the case in object isolation. A simplification of the first part of Gitman and Levine's algorithm, called the Diamond-Function Procedure, was developed. The Subset Uniting Procedure simplifies very easily in the case of a grid, so it was not re-named. The Diamond-Function Procedure eliminates the need to compute and store the lists which are required by Gitman and Levine's algorithm, which gives the Diamond-Function Procedure the property of having fixed storage requirements for a given number of data points. The problem Gitman and Levine's algorithm had with equal-distance points is also eliminated, but the problem of handling points of equal magnitude remains. Equal-magnitude points are handled the same way in both procedures, so the Diamond-Function Procedure inherited the large amount of computation associated with the occurrence of significant numbers of equal-magnitude points in Gitman and Levine's algorithm.

Another procedure, called the Potential Local Maxima Procedure, was developed. This procedure replaces the first part of Gitman and Levine's algorithm, and is applicable to a grid of data points. The second procedure of Gitman and Levine's algorithm is retained, but in

a version which is simplified considerably because the data points lie on a grid. The Potential Local Maxima Procedure has storage requirements which are linear with the number of data points and computational requirements which increase at a rate slightly greater than a linear rate with the number of data points. This procedure, however, generates spurious regions when "flat spots" occur in a scene, and is thus unsuitable for object isolation. The Diamond-Function Procedure and the Potential Local Maxima Procedure are described in Chapter III.

An algorithm which satisfied all four of the above criteria was developed after a similarity between the Subset Uniting Procedure and the Potential Local Maxima Procedure was noticed. This algorithm is called the Unimodal Tree Algorithm. If each object can be made into a unimodal subset by properly enhancing the image, the Unimodal Tree Algorithm will generate one region for each object. The algorithm operates only on data in a grid. Both computational and storage requirements are linear with the number of data points. Points of equal magnitude have no adverse affect on the performance of the algorithm. The Unimodal Tree Algorithm along with some statements about its performance will be presented in Chapter IV.

Experiments using the Unimodal Tree Algorithm are presented in Chapter V. Object isolation was performed on three scenes, two of which contained ellipsoids and spheres in various orientations. One of these scenes contained two touching ellipsoids. The third scene was of two mites. The Unimodal Tree Algorithm was also applied to the prob-

lem of clustering points in a space. Clustering experiments include the Fisher Iris Data [3], bivariate Gaussian clusters, and "ring"-type Gaussian clusters. The Unimodal Tree Algorithm provides a "fast" (computation and storage requirements are linear with the number of data and grid points) method of clustering large numbers of points into clusters of very general shape. Mention is also made of a possible application in optimization of a function of several variables, but no experiments have been carried out in this area.

CHAPTER II

GITMAN AND LEVINE'S FUZZY SET SEPARATION ALGORITHM

2.1 Definitions

Before describing Gitman and Levine's [5] algorithm, it is necessary to define several important terms required to properly understand the algorithm. The first concept needed is that of a fuzzy set. If X is a space of points with elements $x \in X$, then "a fuzzy set A in X is characterized by a membership (characteristic) function $f_A(x)$ which associates with each point in X a real number in the interval [0, 1], with the value $f_A(x)$ representing the 'grade of membership' of x in A". (Zadeh, [9]). There is no need to restrict the value of $f_A(x)$ to the unit interval in either Gitman and Levine's algorithm or in any of the algorithms presented in this thesis. A finite interval of either the real or the integer line may be used instead. All the computer examples done in this thesis use positive integers as the range of the function.

Gitman and Levine denote by μ a point where the maximum function value occurs in A; that is, $f_A(\mu) = \underset{x \in X}{\text{Sup}}\,[f_A(x)]$. A point μ is called the mode of A. In order to define a symmetric fuzzy set and a unimodal fuzzy set, the following two subsets of X are defined:

$\Gamma_{x_i} = \{x \ni f_A(x) \geq f_A(x_i)\}$ and $\Gamma_{x_i, d} = \{x \ni d(\mu, x) \leq d(\mu, x_i)\}$, where

$x_i \in X$ and $d(u, v)$ is a metric, or distance measure, between points u and v; Γ_{x_i} is the set of points in X each of which have a function value greater than the function value of x_i; $\Gamma_{x_i,d}$ is the set of points in X having distance from μ not greater than the distance from μ to x_i. Note that the terms "symmetric set" and "unimodal set" will be used instead of "symmetric fuzzy set" and "unimodal fuzzy set" when the context is clear.

Definition: A fuzzy set A is <u>symmetric</u> if and only if, for every point $x_i \in X$, $\Gamma_{x_i} = \Gamma_{x_i,d}$. This definition says that moving closer to the mode and increasing in function value are synonymous in a symmetric fuzzy set. (See Figure 3)

Definition: A fuzzy set A is <u>unimodal</u> if and only if the set Γ_{x_i} is connected for all $x_i \in X$. If a fuzzy set is split up into region(s) having a function value greater than α and at most one region results for all α in the range of the function associated with the fuzzy set, then the fuzzy set is unimodal, and visa-versa. (See Figure 4)

When working with a computational algorithm, it is necessary to have a finite number of points in a discrete space. Gitman and Levine denote a sample of N points from A by $S = \{(x_i, f_i)^N\}$, where $x_i \in X$ and f_i is the function value corresponding to x_i. A partition of S into m subsets, each with maximum function value μ_i, is denoted by $\{(S_i, \mu_i)^m\}$. For a point $x_k \in S_i$, let x_t be defined by

$d(x_t, x_k) = \min_{x_j \in (S - S_i)} [d(x_k, x_j)]$; x_t is the closest point to x_k not in S_i.

Figure 3: Symmetric Fuzzy Sets

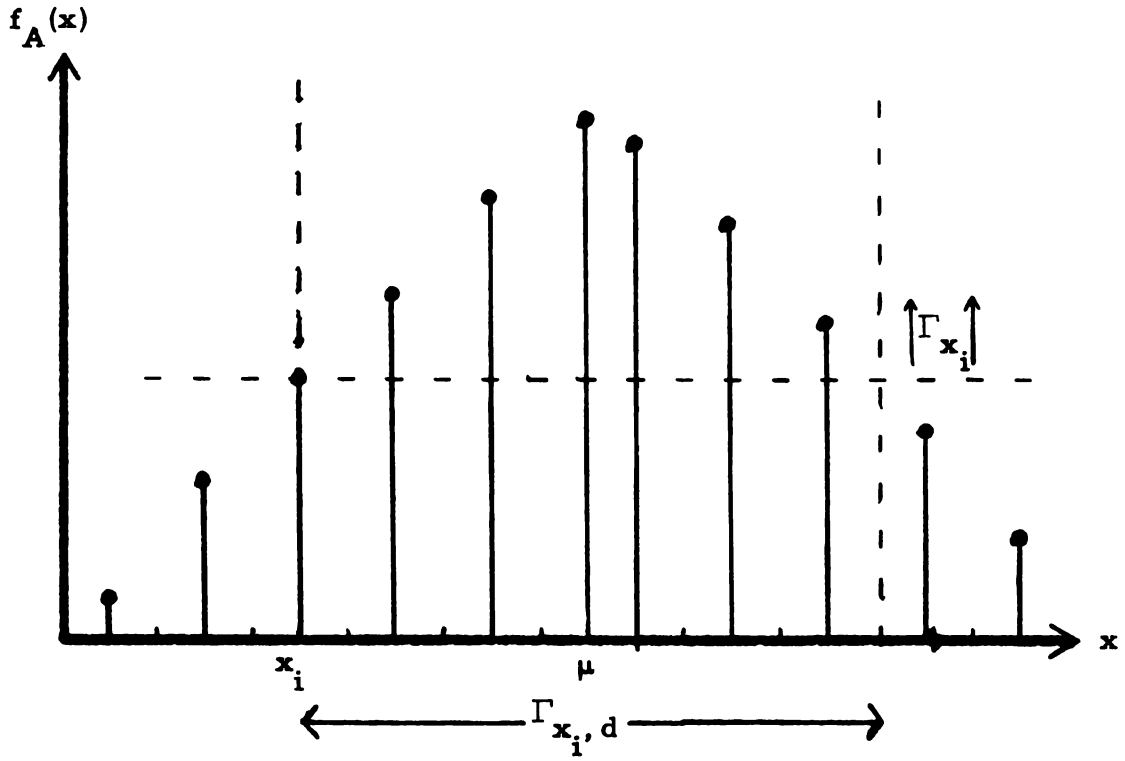


Figure 3A. A Symmetric Set. $\Gamma_{x_i} = \Gamma_{x_i, d}$

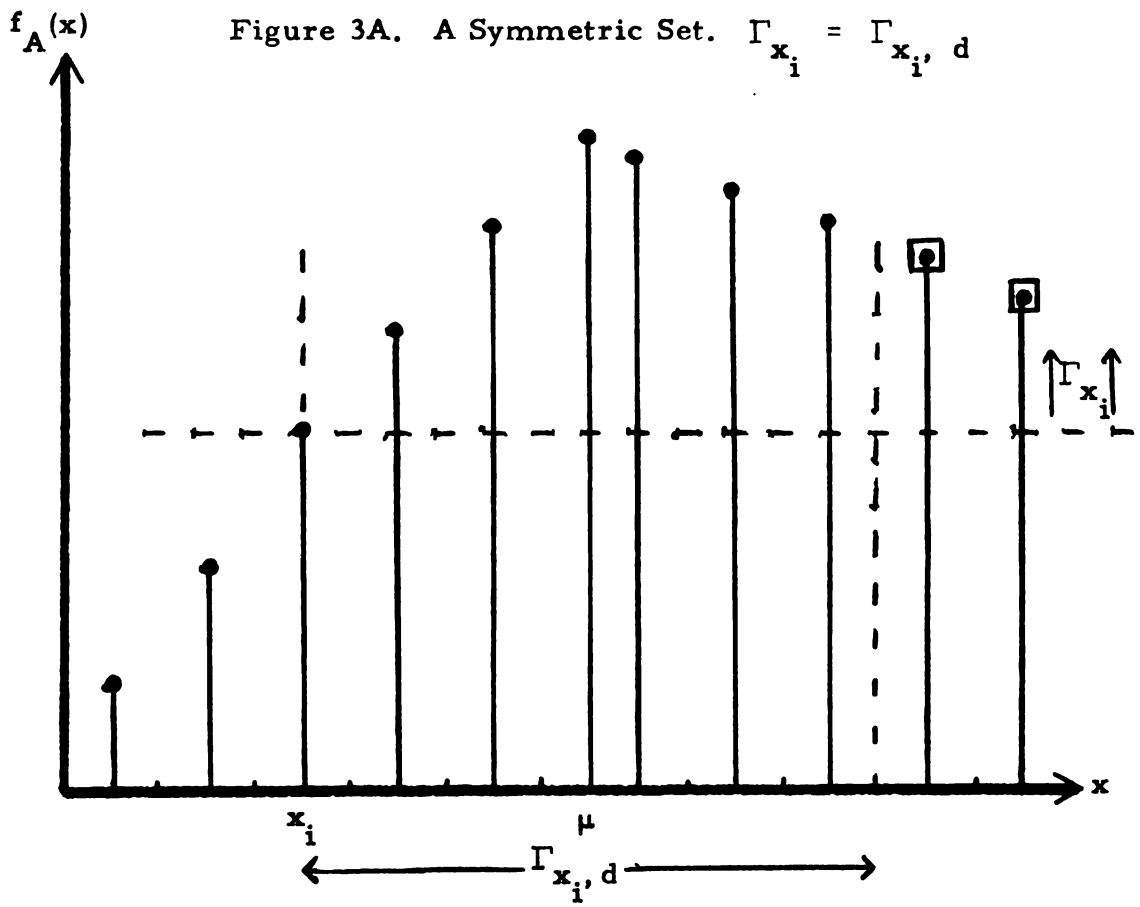


Figure 3B. Non-Symmetric Set. $\Gamma_{x_i} \neq \Gamma_{x_i, d}$, because the "boxed" points are in Γ_{x_i} but not in $\Gamma_{x_i, d}$.

Figure 4. Unimodal Fuzzy Sets

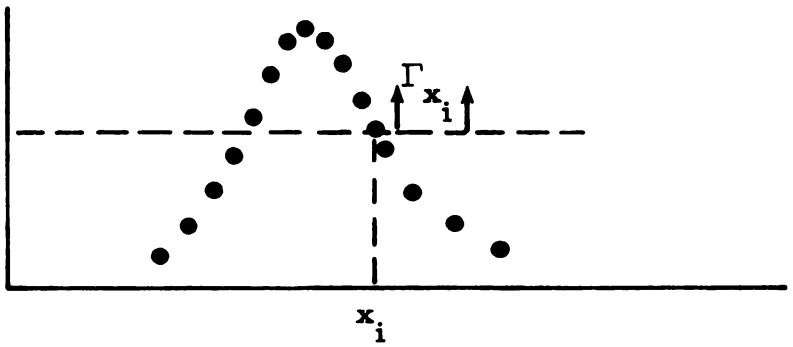


Figure 4A. A Unimodal Fuzzy Set. Γ_{x_i} is connected.

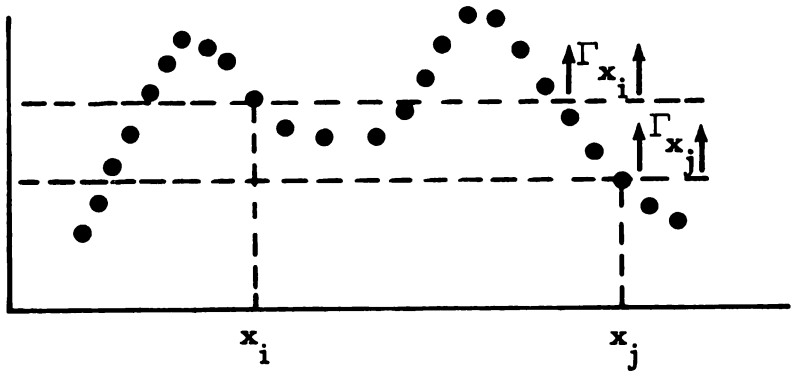


Figure 4B. Non-Unimodal Fuzzy Set. Γ_{x_i} is in two "pieces". Note the necessity for checking the connectedness of Γ_{x_i} for every x_i. Γ_{x_j} is connected, but the set is not unimodal.

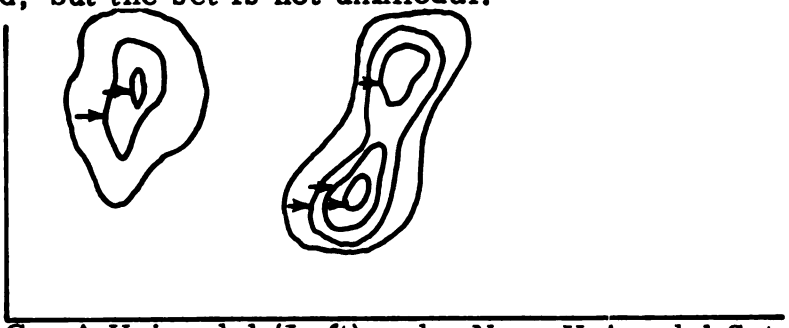


Figure 4C. A Unimodal (Left) and a Non-Unimodal Set in two dimensions shown by contour lines. The small arrows indicate the contour at the head of the arrow has a higher value than the contour at the tail of the arrow.

Definition: x_k is an interior point of S_i if the set $\{x \ni d(x_t, x) < d(x_t, x_k)\}$ includes at least one element of S_i. In other words, x_k is an interior point of S_i if there is another point in S_i closer to x_t, the point closest to x_k but not in S_i. (See Figure 5)

Note that there is an element of ambiguity in this definition of interior point. If two or more points outside S_i are found to have the same minimum distance to x_k, one of them might satisfy the criterion to make x_k an interior point and the other one not. For example, consider the four points shown in Figure 6 with the indicated partition into S_1 and S_2. The distance between x_{t_1} and x_k is the same as the distance between x_{t_2} and x_k, and is the minimum distance to x_k for $x \notin S_1$. x_ℓ is closer to x_{t_1} than x_k is to x_{t_1}, making x_k an interior point by definition. x_ℓ is not closer to x_{t_2} than x_k is, however, which makes x_k not an interior point of S_1. Since a scene is ordinarily scanned on some sort of a uniformly-spaced grid, some method of deciding if a point is to be considered interior or not interior in such a situation will be needed. This problem will be discussed later. Gitman and Levine do not treat this situation.

2.2 Gitman and Levine's Algorithm

The algorithm developed by Gitman and Levine consists of two parts: procedure F and procedure S. (See Figure 7) In procedure F, the local maxima of the fuzzy set are found. A local maximum in the continuum becomes a mode μ_i of subset S_i, subject to certain criteria

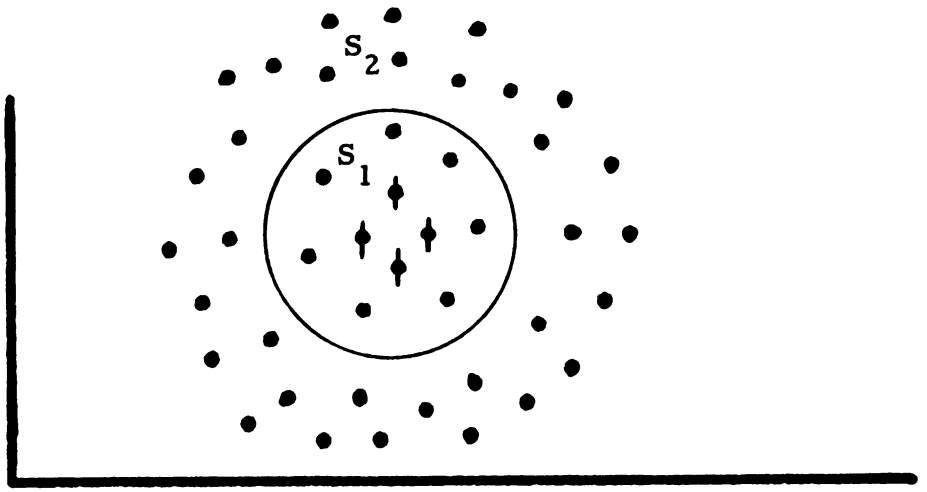


Figure 5. Interior Points

The interior points of S_1 are "barred". The nearest point in S_2 to each "barred" point has another point in S_1 closer to it than the "barred" point. The "unbarred" points in S_1 are not interior points because they do not satisfy the definition of interior point.

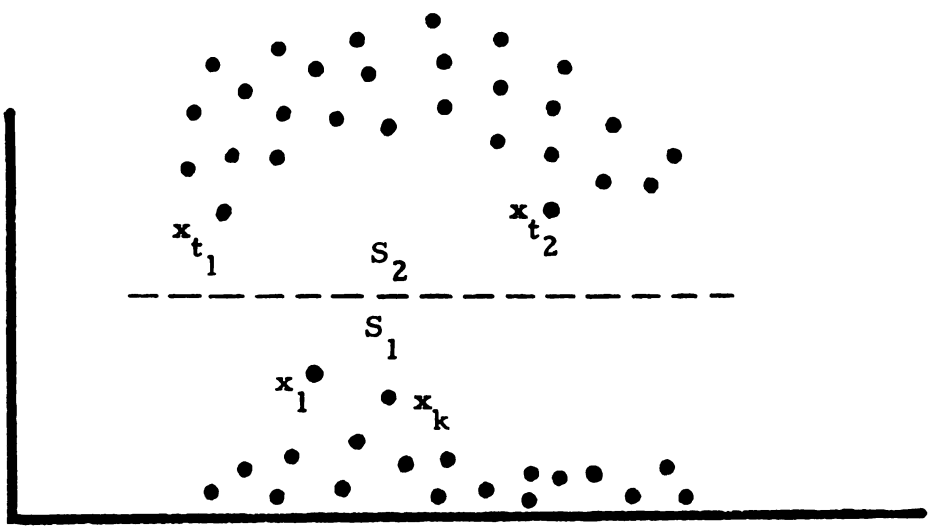
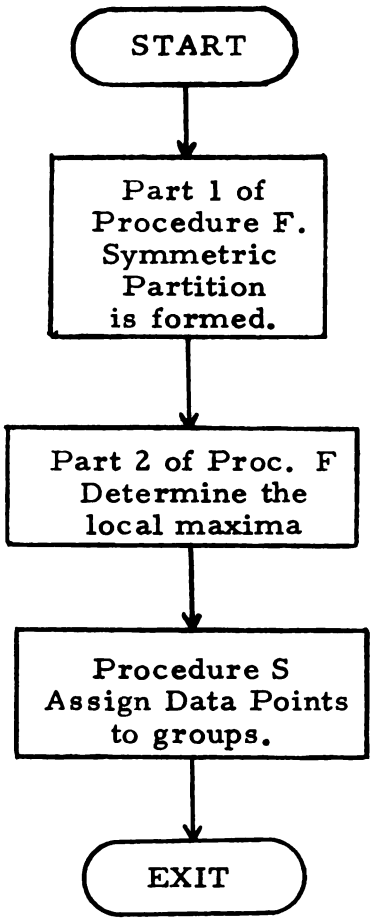


Figure 6. Ambiguity in Definition of Interior Points. Is x_k interior?

Figure 7. Flowchart of Gitman and Levine's Algorithm



on the sampling used (theorem 1 in Gitman and Levine). All of the S_i's generated by procedure F are symmetric; μ_i is a local maximum if μ_i is an interior point of S_i. The criteria on the sampling reflect the fact that there must be a symmetric subset of radius ϵ about each local maximum in the continuum, that the points in the sampling must be spaced no further apart than ϵ, and that local maxima must be sample points. In general, none of these conditions are known to hold, but they should be fairly well approximated for reasonable samples. Procedure S will be discussed later.

2.2.1 Part One of Procedure F

Part one of procedure F requires generating a sequence $A_o = (y_1^0, y_2^0, \ldots)$ of the points of the sample S in decreasing order of magnitude (assume for now that there are no points of equal magnitude). (See Figure 8). y_1^0 is the global maximum of the sample, y_2^0 is the next higher point, and so forth. Another sequence $A_1 = (y_1^1, y_2^1, y_3^1, \ldots)$ of points in S ordered by increasing distance from y_1^0 is generated (assume for now that no pair of points is the same distance apart as any other pair). The points in A_o shall all be inspected one at a time, in order, and assigned to a symmetric subset, or group. Assign the points of A_o, in order, to group 1 until some r is found such that $y_r^0 \neq y_r^1$, and $y_i^0 \equiv y_i^1$ for $0 < i < r$. (The notation "$\equiv$" means here "is identically the same point"). When this situation occurs, the next point down in magnitude (y_r^0) is not the same as the next point out in distance (y_r^1). Group one might not remain symmetric if y_r^0 was assigned to it, since there

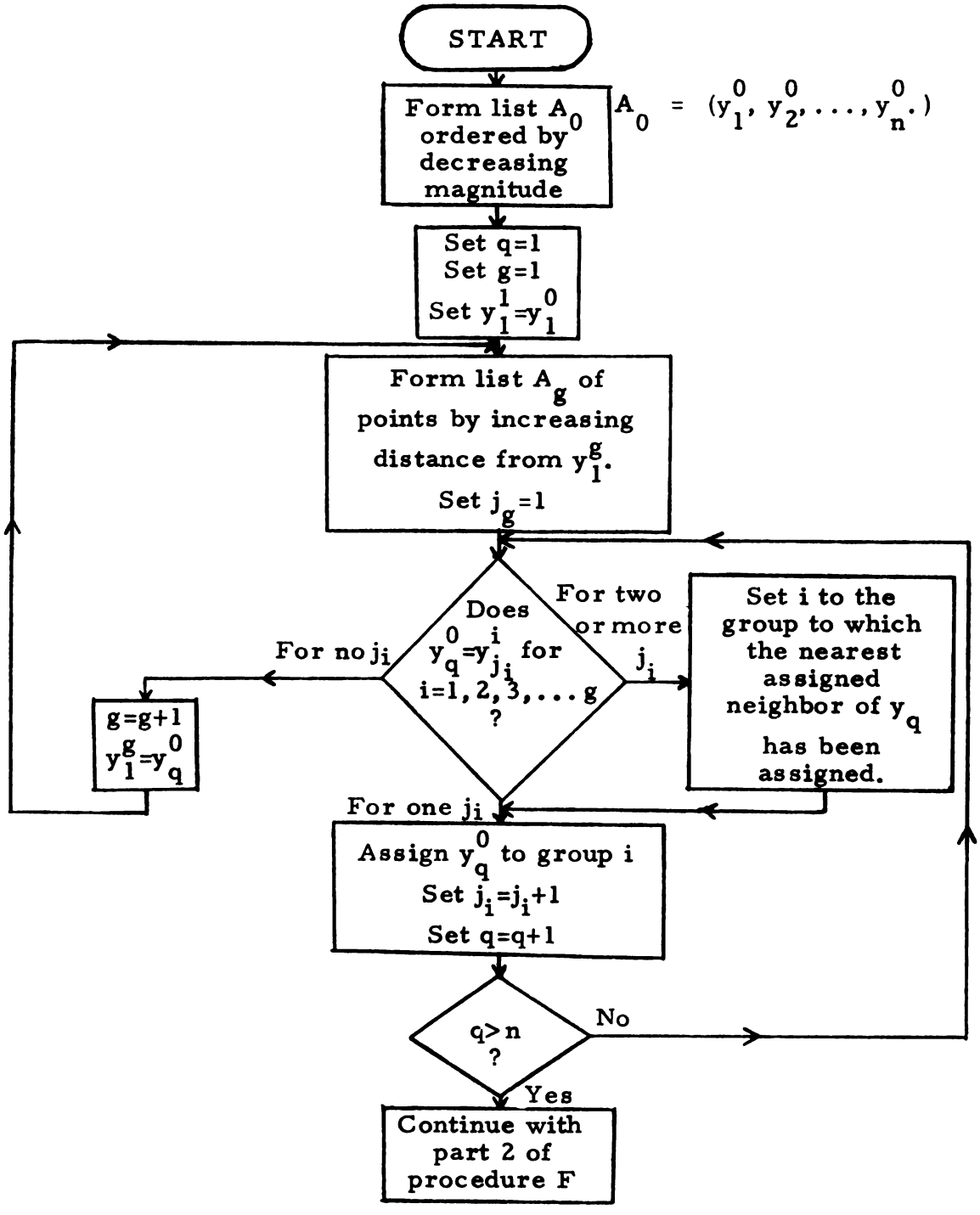


Figure 8. Flowchart of Part 1 of procedure F. There are N data points. This flowchart does not contain provision for equal-magnitude and equal-distance points.

is a closer point of lower magnitude. At this point group two is initiated by setting $y_1^2 = y_r^0$ and forming the sequence A_2 of points ordered by increasing distance from y_1^2. The list A_2 may be terminated when a point already assigned (y_i^0, $0 < i < r$) is encountered. Future assignment to group two would halt there anyway since the point can never be found in list A_o again.

Now suppose that g groups have been initiated in the same manner as groups one and two were initiated above. There are g sequences $A_1, A_2, \ldots, A_g$, each of which has a candidate for assignment. These candidates are each denoted by y_c^i, where i is the group number, $0 < i \leq g$. Suppose y_q^0 is the current point to be assigned in the A_o list (points y_i^0 for $0 < i < q$ have already been assigned). There are three cases:

i) No identity between y_q^0 and y_c^i for any group.

ii) Equality between y_q^0 and y_c^i for exactly one group.

iii) Equality for more than one group.

The action to be taken for each of these cases is as follows:

Case i): Form group g + 1 with mode $y_1^{g+1} = y_q^0$ and the sequence A_{g+1}. A_{g+1} consists of the points of the sample S in increasing distance order from y_1^{g+1}. Increment g and q, determine which case (i, ii, or iii) is present, and continue.

Case ii): Assign y_q^0 to the matching group, increment q, and continue.

Case iii): Assign y_q^0 to the matching group containing the assigned point closest to y_q^0.

For an example of procedure F on a simple one-dimensional data set, see Gitman and Levine [5]. This procedure will always generate symmetric subsets. If, however, ambiguity enters into either the ordering of the points by magnitude or by distance, or by both, the procedure may generate many more symmetric subsets than are really necessary. To see why this happens, consider the case of three equally-spaced points of different magnitude. (See Figure 9). The points will be ordered by magnitude correctly, but when the points are subsequently ordered by distance, the second and third points could be placed either way. One way will match up with the ordering in A_o and the other way will not. If the lists do not match, another group will be formed, and the distance ordering for this group will also be ambiguous. If one choice is made, the mode of group one will be found, and the A_2 list will be terminated. If the other choice is made, the remaining point will become the next candidate for group two. This point will be assignable to either group one or two, but the rule for case iii) will not tell which. In such a case, assignment is arbitrary. Part one of procedure F may generate either one or two symmetric subsets in this simple example. In addition the procedure will enable the generation of a minimum number of symmetric subsets in the case of distance equality. Simply re-order all the equal-distance points in each of the lists A_1, $A_2, \ldots,$ so that the order of these points is the same as their order in the A_o list.

Points having equal magnitude also cause the generation of more

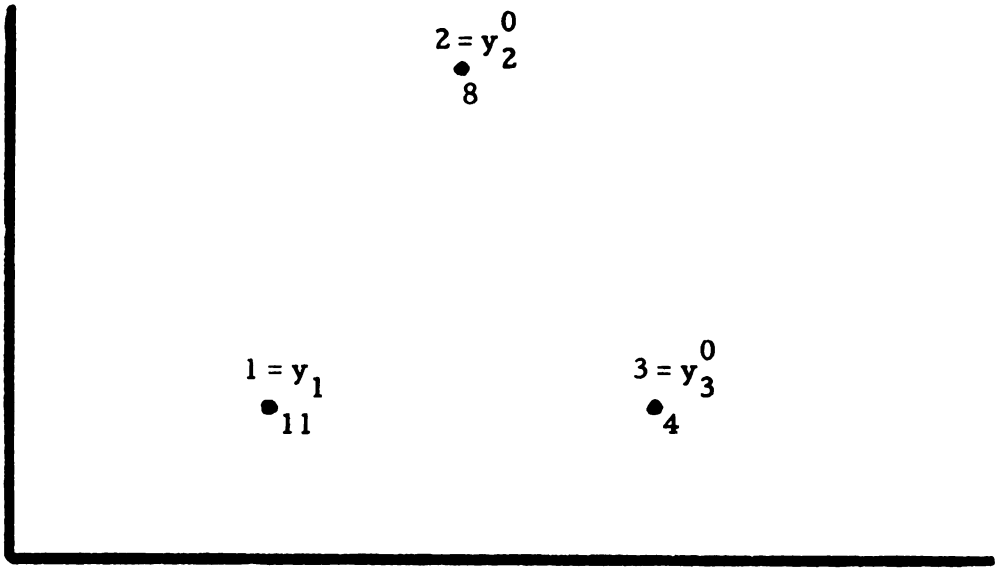


Figure 9. The Problem With Equal-Distance Points

$A_0 = (1, 2, 3)$

A_1 may be arbitrarily formed in either of two ways:

1. $A_1 = (1, 2, 3)$

$y_1^0 = y_1^1 \Rightarrow 1 \rightarrow$ group 1

$y_2^0 = y_2^1 \Rightarrow 2 \rightarrow$ group 1

$y_3^0 = y_3^1 \Rightarrow 3 \rightarrow$ group 1

2. $A_1' = (1, 3, 2)$

$y_1^0 = y_1^1 \Rightarrow 1 \rightarrow$ group 1

$y_2^0 \neq y_2^1 \Rightarrow$ form group 2. A_2 may now be formed as either:

$A_2 = (2, 1, 3)$ or $A_2' = (2, 3, 1)$

$y_2^0 = y_1^2 \Rightarrow 2 \rightarrow$ group 2

$y_3^0 = y_2^1 \Rightarrow 3 \rightarrow$ group 1

$y_2^0 = y_2^1 \Rightarrow 2 \rightarrow$ group 2

$y_3^0 = y_2^1$ and $y_3^0 = y_2^2 \Rightarrow 3 \rightarrow$ either group 1 or group 2.

(Assignment to group 1 or group 2 is arbitrary since the distance to the nearest assigned neighbor is a tie.)

symmetric subsets than necessary. For example, consider three points having the same function value, but separated by different distances. (See Figure 10) An arbitrary ordering is selected for the A_o list, and the A_1 list is generated unambiguously. The second element of the two lists may match or not match, arbitrarily. Either one or two symmetric subsets may result.

It is desirable to generate a small number of symmetric subsets for three reasons. First, the initiation of each subset requires the computation and ordering of distances between the mode of the new subset and some of the data points (data points at a distance further than the minimum distance from the mode to points already assigned need not be considered). Minimizing the number of subsets will reduce this computation. Secondly, each time a point is assigned to a group, all the groups must be searched for a candidate point identical to the point being assigned. Reduction of the number of groups will also reduce the computation for this search. A third reason is that a local maximum may be missed if the symmetric set around it is terminated prematurely.

A modification of case i) may be made which enables the procedure to generate a smaller number of symmetric subsets in the case of equality in function values. If no matching points are formed as in case i), continue the search using the points which have the same magnitude as the first point. Initiate a new group only when none of these equal-magnitude points satisfy either case ii) or case iii). Note that if equal distance points are present, the equal-distance points in each of

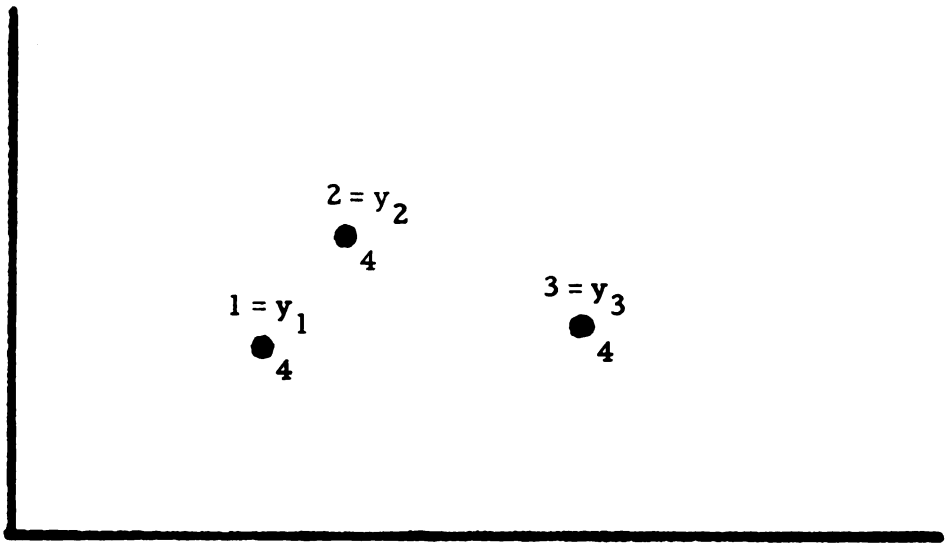


Figure 10. The Problem with Equal-Magnitude Points.

Six cases exist for the possible orderings of A_0.

A_0 = (1, 2, 3)	A_0 = (1, 3, 2)	A_0 = (2, 1, 3)
A_1 = (1, 2, 3)	A_1 = (1, 2, 3)	A_1 = (2, 1, 3)
1→group 1	1→group 1	2→group 1
2→group 1	A_2 = (3, 2, 1)	1→group 1
3→group 1	3→group 2	3→group 1
	2→group 1	
A_0 = (2, 3, 1)	A_0 = (3, 1, 2)	A_0 = (3, 2, 1)
A_1 = (2, 1, 3)	A_1 = (3, 2, 1)	A_1 = (3, 2, 1)
2→group 1	3→group 1	3→group 1
A_2 = (3, 2, 1)	A_2 = (1, 2, 3)	2→group 1
3→group 2	1→group 2	1→group 1
1→group 1	2→group 2	

the A_i lists will have to be either re-ordered or searched for equality during this procedure.

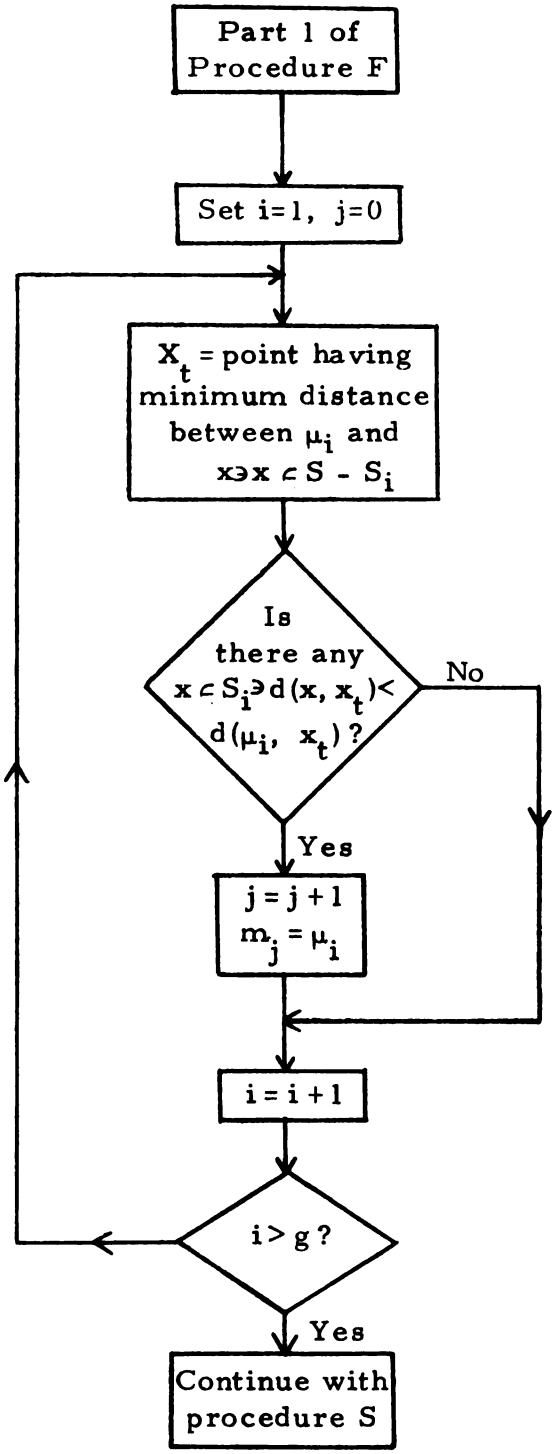
2.2.2 Part Two of Procedure F

Part two of Procedure F uses the symmetric partition generated by part one as input and determines the local maxima of the sample function. (See Figure 11). The procedure is as follows: if μ_i is an interior point of S_i, then μ_i is a local maximum; otherwise, it is not. This test is performed for each S_i in the symmetric partition. In other words, determine the minimum distance from μ_i to the points not in S_i. Call the point where this minimum distance occurs x_{t_i}, and assume x_{t_i} is unique for now. μ_i is a local maximum if there is a point in S_i which has distance from x_{t_i} less than the distance from μ_i to x_{t_i}. Otherwise, μ_i is not an interior point of S_i, and hence is not a local maximum.

Note that part two of Procedure F has two inherent problems. First is the problem in the definition of interior point which was mentioned earlier. The problem of a non-unique x_{t_i} may be resolved three ways: 1) the mode is interior only if all x_{t_i} satisfy the condition, 2) the mode is interior if any x_{t_i} satisfies the condition, or 3) the mode is interior if an arbitrary one of the x_{t_i} satisfies the condition for the mode to be an interior point. The third scheme was used in the implementation of Gitman and Levine's algorithm which was used for this thesis.

The second problem associated with part two of Procedure F is the occurrence of non-unique μ_i in S_i for one or more groups. The problem

Figure 11. Part 2 of Procedure F.



arises because an arbitrary one of these equal-magnitude points was selected as the mode of the symmetric set. If this mode is surrounded by points of equal magnitude, it will be classified as a local maximum even if this flat region is not a local maximum at all. An example of such a flat spot is a level region on the side of a hill, as shown in Figure 12. To prevent such a flat region from generating any local maxima Gitman and Levine offered the following suggestion: "To solve this problem, we have modified part two of procedure F (in which a search for the local maxima is performed) in the following way. Let S_{ii} be the subset of points in S_i which have the same (maximal) grade of membership as μ_i; then every point in S_{ii} is examined as the mode of S_i. If at least one of these points is on the boundary of S_i, then μ_i is not considered as a local maximum." Many counter-examples exist to this solution. Consider the case of an elliptical flat region in two-dimensional Euclidean space surrounded by lower values decreasing with increasing distance from the edge of the flat region. (See Figure 13) Part one of procedure F will pick an arbitrary one of these points in the flat spot as the global maximum and start the first symmetric group with it. The first group will be extended until a point in the distance list A_1 but not in the flat region is encountered. Another group will start on the flat region at a point of the flat region not in the first group. Such a point will exist because there will be points in the ellipse not included in the disk-shaped symmetric first group. This group will grow until it hits either the edge of the flat region or the first group,

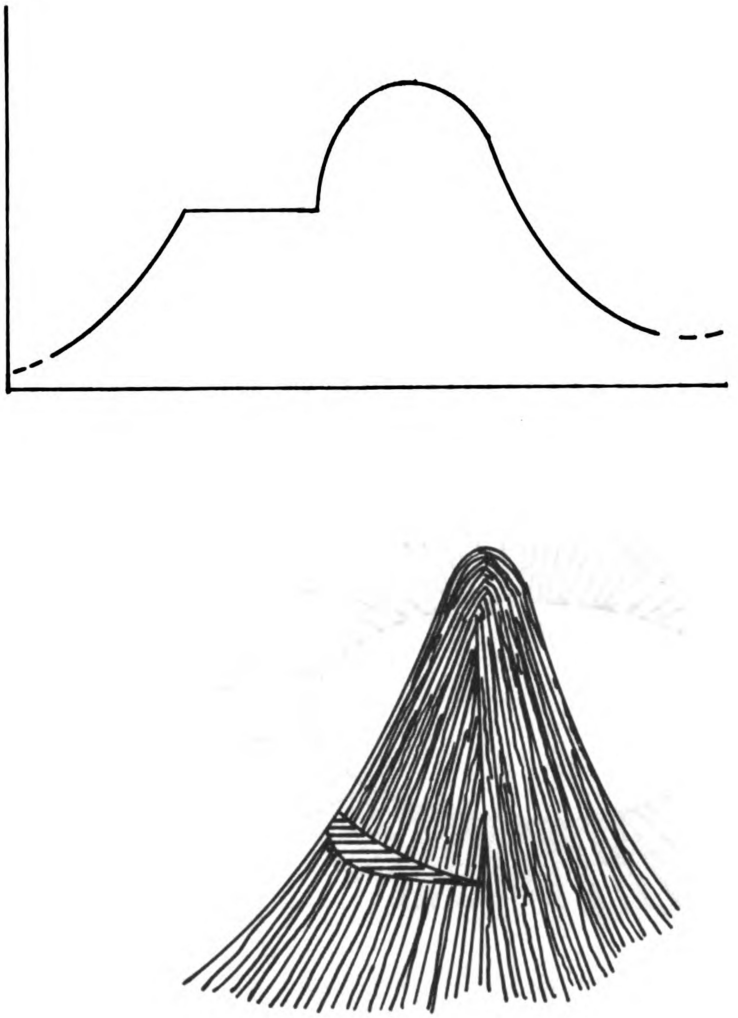


Figure 12. Flat Regions Which are not Local Maxima

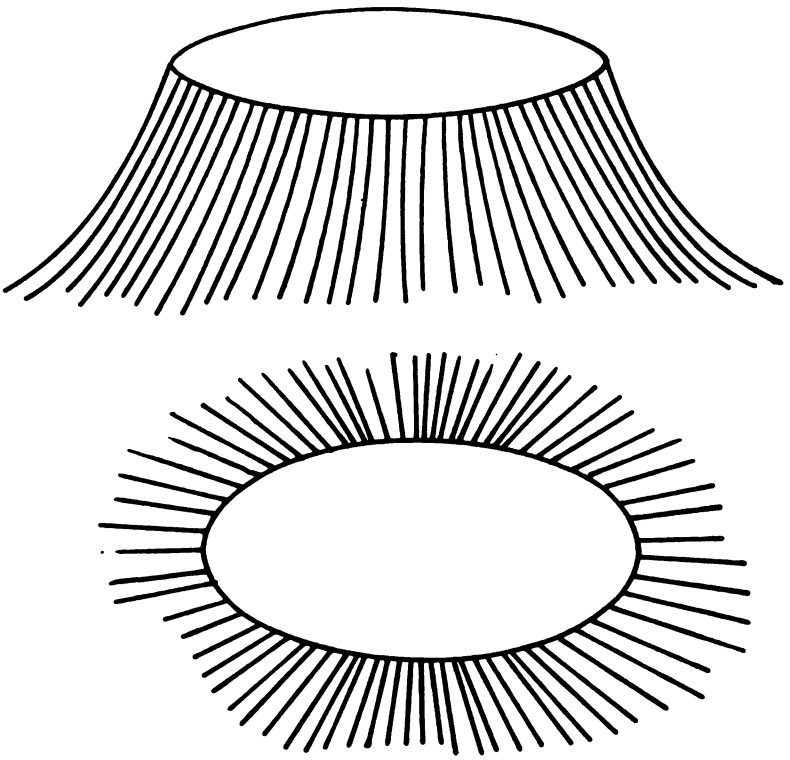


Figure 13. An Elliptical Flat Region Surrounded by Lower-valued Points.

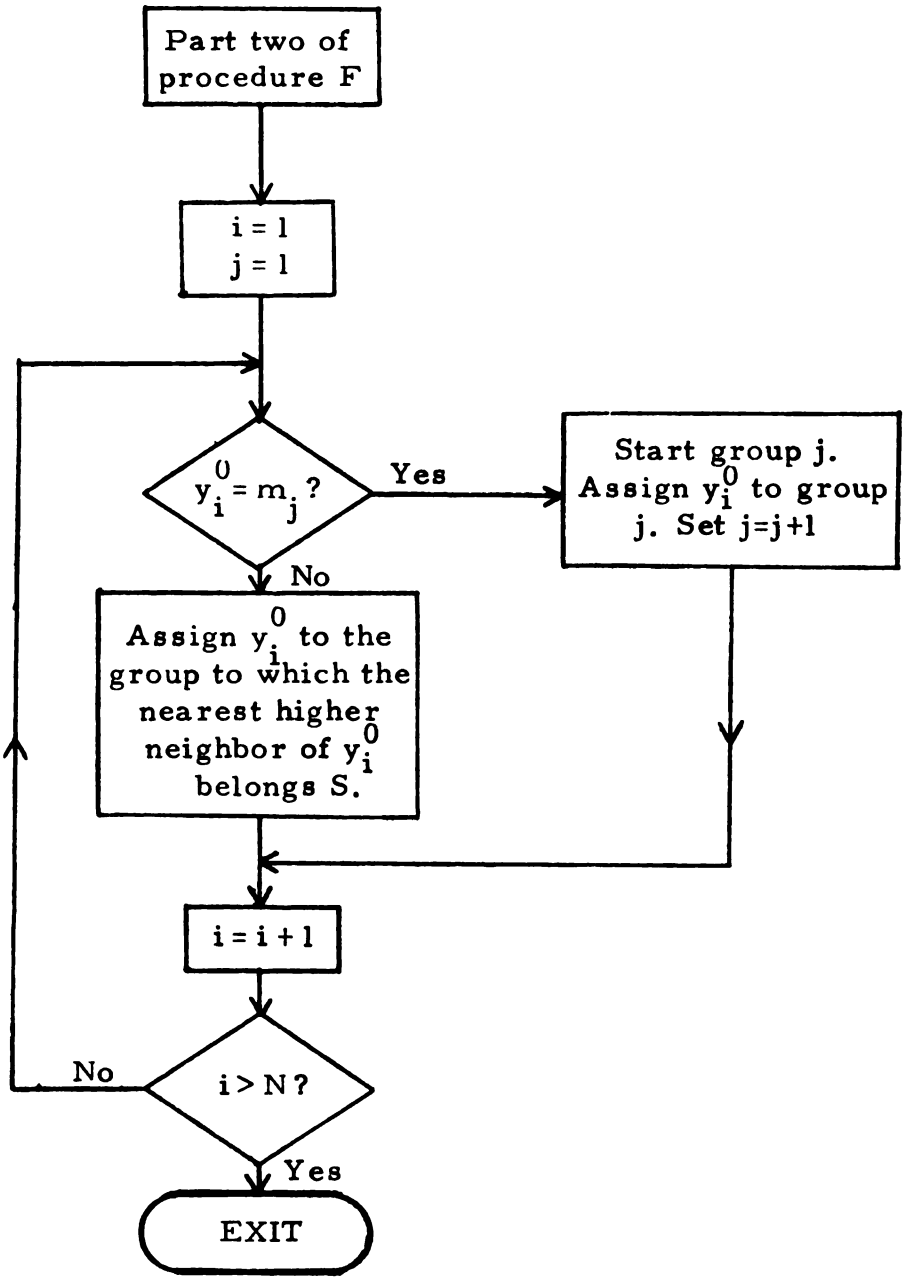
whereupon another group will be started. This process will continue until the flat region is covered by symmetric groups. Group one and other groups which hit the edge of the flat spot may be extended by a point or two, and new groups will be initiated to cover the sides of the "butte". Now consider the first subset. There will be points along its edge which are in the flat spot but are not interior points. The same observation will hold for boundary points in each group having points in the flat region. The result is that this region will be missed as a local maximum. The only way this flat region could be kept as a local maximum would be to include the entire flat region and some of the surrounding area in a symmetric subset.

In object isolation the problem with missing flat local maxima described above is intolerable. The intensity function is normally quantized, perhaps into only a few levels. Occurrences of flat regions which are local maxima will be a common situation in such images. A procedure for correctly handling the case of non-unique modal values is mandatory if object isolation is to be performed. Such a procedure will be discussed after procedure S is described.

2.2.3 Procedure S

Using the local maxima generated by procedure F as input, procedure S partitions the sample S into unimodal regions. Points which are not local maxima are assigned in the order in which they appear in sequence A_o. (See Figure 14). Each local maximum will start a new unimodal group. The procedure is as follows: Assign the point y_j^0 (in

Figure 14. Flowchart of Procedure S. M_j is the j^{th} local maximum.



location j of sequence A_o) to the group in which its nearest neighbor with a higher function value has been assigned, except for the local maxima. When a local maximum is encountered, start a new group. "Higher function value" and "previously assigned" are synonymous in this procedure because of the decreasing-magnitude order of assignment.

Gitman and Levine prove a theorem which states that procedure S may be replaced by a procedure which unites the symmetric subsets to form unimodal groups if the maximum diameter of a disk in the continuum containing no sample points shrinks to zero. (Theorem three in Gitman and Levine). Furthermore, they point out that if this uniting procedure is used, procedure S reduces to an automatic classification of the points, providing that for each mode the nearest point with a higher grade of membership is recorded during part one of procedure F. The procedure involved is to assign all the points in each subset whose mode is not a local maxima to the group to which the nearest higher neighbor to the mode of this subset was assigned. Assignment is done by decreasing order of modal value for all the subsets. The points in each subset containing a local maximum start a new group. This procedure suggests a similar procedure which replaces part two of procedure F and procedure S.

2.3 The Subset Uniting Procedure

Since the data points can be assigned by selectively uniting symmetric subsets, it should be possible to obtain a similar assignment of the points by using a different rule for uniting these subsets. A requirement

for uniting two subsets is that the subsets be "adjacent" to one another. A definition of adjacent symmetric subsets which satisfies the idea of adjacency follows.

Definition: Let z_{ij} be the closest point (in case of ties in closeness, the highest of the closest points) in S_i to any point in S_j, and let z_{ji} be the closest point (or the highest of the closest points) in S_j to any point in S_i, where S_i and S_j are symmetric subsets. S_i and S_j are <u>adjacent</u> if

$$d(z_{ij}, z_{ji}) \leq \max_{w \in S - (S_i \cup S_j)} [d(z_{ij}, w), d(z_{ji}, w)], \text{ where } d(u, v) \text{ is a}$$

metric function between points u and v. (See Figure 15). This inequality has been termed the ultrametric inequality (Johnson [7]).

Assume $i < j$. Then $\mu_i \geq \mu_j$ from part one of procedure F. The following conditions must hold for S_i and S_j to be united in order to maintain a unimodal set as the result of this union and of previous unions which may have been made using S_i and S_j, and the subsets united to them:

a) S_i and S_j must be adjacent.

b) z_{ji} and μ_j must have the same value.

c) The value of z_{ij} must be no less than that of z_{ji}.

d) μ_i and μ_j must have magnitude not less than the magnitude of the mode of any of the subsets with which S_i or S_j, respectively, has been united with in the past.

The order in which these subset pairs are checked to see if they

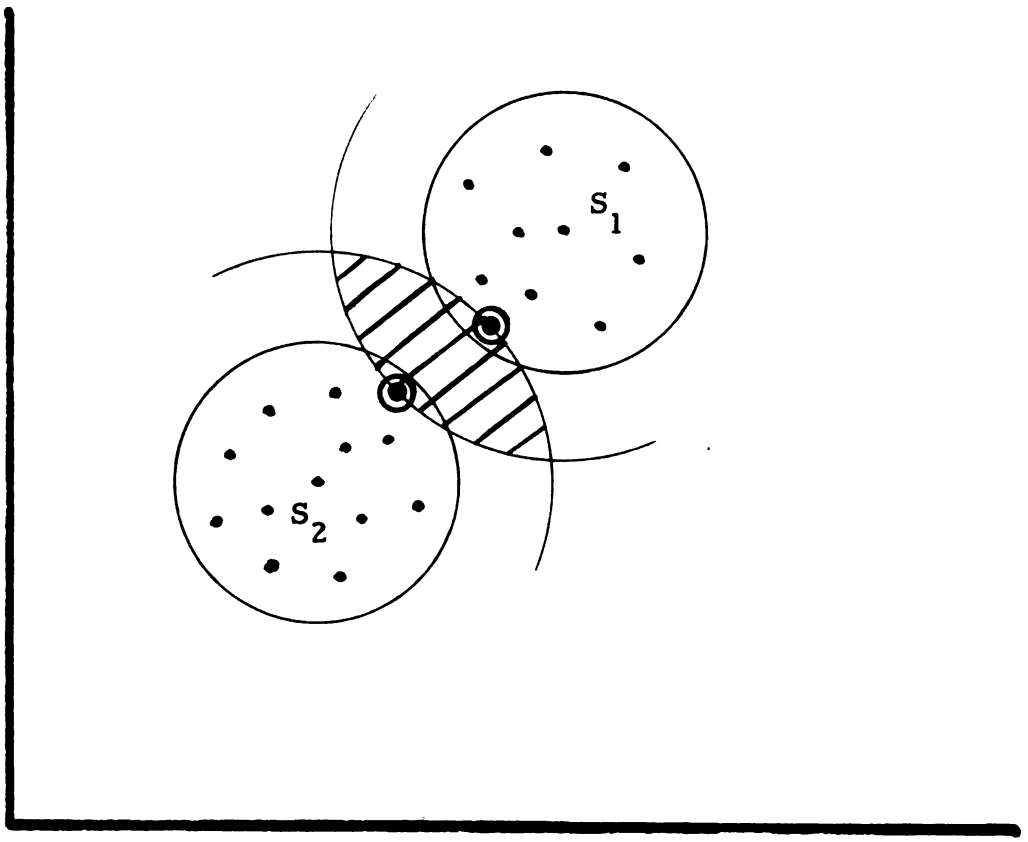


Figure 15. Illustration of Adjacency

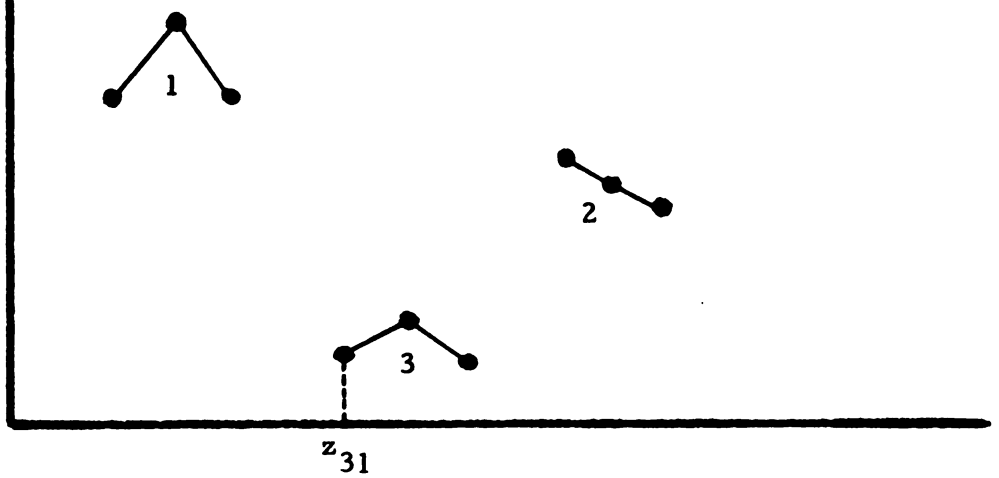
The circled points are the points in each symmetric subset S_1 and S_2 closest to the mode of the other subset of the pair. If no points in $S - (S_1 \cup S_2)$ fall in the shaded region, S_1 and S_2 are adjacent. Otherwise, S_1 and S_2 are not adjacent.

can be united is as follows: First try the pair (S_1, S_2), then (S_1, S_3), (S_1, S_4), and so forth to (S_1, S_g). Then (S_2, S_3) are checked, then (S_2, S_4), and so forth, until the pair (S_{g-1}, S_g) has been checked. The fact that checks b), c), and d) above are needed is verified in Figure 16. When groups are united, it is not necessary to recall all the previous unions these groups have experienced, as requirement d) would indicate. Rather, a flag may be associated with each subset, and set if its subset is united with another subset having a higher modal value. Two subsets may not be united if both flags are set.

The above procedure will be termed the "Subset Uniting Procedure". Note that since a single point is a symmetric subset, the Subset Uniting Procedure is applicable to the data points individually before part one of procedure F separates them into symmetric subsets. The Subset Uniting Procedure requires $\frac{g(g-1)}{2}$ adjacency tests for g subsets. If each point is considered as a subset in this procedure, and there are n points, $\frac{n(n-1)}{2}$ adjacency tests are performed. If each symmetric subset resulting from part one of procedure F has about k points in it, then $g \cong n/k$. The ratio of the number of adjacency tests necessary if procedure F is not used to the number required if procedure F is used is then about $\frac{n(n-1)}{\frac{n}{k}(\frac{n}{k} - 1)}$ or $k^2\frac{(n-1)}{(n-k)}$, or approximately k^2 for n much larger than k. Use of part one of procedure F will thus reduce the number of adjacency tests required for the Subset Uniting Procedure. Each adjacency test requires substituting each of the points not in either subset

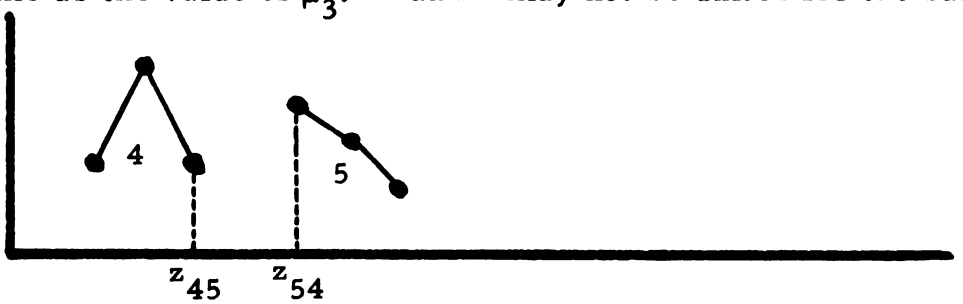
Figure 16. Illustration of the Conditions of the Subset Uniting Procedure

Each connected set of dots is a symmetric subset. Subsets are numbered by decreasing value of their mode. Dashed lines indicate previously united subsets.

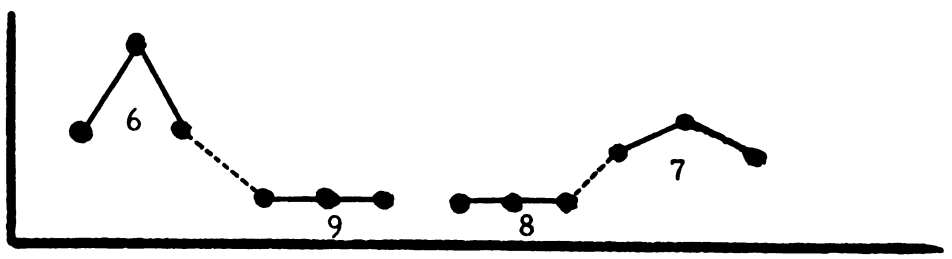


Case a). 1 and 2 may not be united because they are not adjacent. If they were united, the result would not be unimodal because the union of 1 and 2 is not connected.

Case b). 1 and 3 may not be united because the value of z_{31} is not the same as the value of μ_3. 2 and 3 may not be united for the same reason.



Case c). 4 and 5 may not be united because $z_{45} < z_{54}$.



Case d). 6 is united with 9 and 7 is united with 8. If 8 and 9 are united (they satisfy conditions a), b), and c)), the result is not unimodal.

being tested into the ultrametric inequality until either a violation of adjacency is found or all the points are checked. The amount of computation involved in an adjacency test is relatively independent of the size of the two subsets. About n - 2k substitutions into the ultra-metric inequality are required if procedure F is used, and n-2 are required if procedure F is not used, in the "worst case" where the subsets are adjacent. In the case of a pair of subsets, each of size k, the closest point in each subset to the mode of the other must be found. This operation will require 2k comparisons. These 2k comparisons may be considered to roughly "make up for" the 2k-2 substitutions into the ultra-metric inequality saved when the two k-point subsets are tested for adjacency. Thus, using part one of procedure F to generate subsets containing about k points apiece will reduce the computation involved in the Subset Uniting Procedure by about a factor of k^2.

The amount of computation involved in part one of procedure F can vary quite a bit depending upon the number of points of equal magnitude which occur. Two major computational operations are involved. First is the ordering of points by magnitude, which occurs only once, and the ordering of points by distance from each mode, which occurs g times. An ordering algorithm called TREEUP was used [2, 4]. Bertziss [2] claims the worst case computation requirements for TREEUP are approximately $2N(\log_2 N - 1)$ comparisons and $N(\log_2 N - 1)$ interchanges. g + 1 orderings must be made, but not all of them involve all n data points. Additional ordering is necessary if points of equal distance to

some modes are present.

The second computational operation involved in part one of procedure F is the actual assignment of points. If points of equal magnitude are not present, g comparisons of y_q (the point to be assigned) and y_c^i (the current candidate point for each group) must be made. Furthermore, the closest neighbor point which has already been assigned must be found if more than one match exists between y_q and the y_c^i. In the case of equal-magnitude function values, many successive points may have to be checked before assignment can be made or a new group can be initiated. For example, consider thirty points of equal magnitude and twenty groups. If y_q is the first of these thirty points and y_q cannot be assigned to any of the twenty groups, then 600 comparisons will have to be made. If the next point cannot be assigned to any of the twenty old groups or the group just formed from the first of the equal-magnitude points, 609 more comparisons will be made, and so forth.

The storage requirements of part one of procedure F are not fixed for a given number of points. In addition to the storage required for the function value of each point is the storage for lists $A_0, A_1, \ldots, A_g$. The lengths of A_0 and A_1 are fixed at n, but the lengths of $A_2, A_3, \ldots, A_g$ vary depending upon the number of points which can possibly be included in each subset. Gitman and Levine state that 5n storage locations were usually sufficient for the lists $A_0, A_1, A_2, \ldots,$ and A_g in their work. In addition, auxiliary working storage arrays are required during ordering and to hold computed distances. 7n locations were found to be

sufficient for these arrays.

The computational requirements of part two of procedure F and of procedure S will not be discussed because local maxima may be missed using these procedures. The use of part one of Procedure F and the Subset Uniting Procedure requires so much computation for the large number of points encountered in an object isolation problem that the use of these procedures for object isolation is impractical on even the fastest computers.

Several special functions defined on a 10 x 10 grid were run on the M.S.U. CDC 6500 computer using FORTRAN EXTENDED for a programming language. These 10 x 10 functions included a crescent-shaped unimodal region and several simpler cases (see Figure 17). The average execution time for these 10 x 10 scenes was about 5 central-processor seconds. When the number of data points was expanded to 484 (a 22 x 22 scene of two mites; see Figure 18), the central processor time increased to 197 seconds. All the groups generated in each trial case were unimodal.

The cost associated with using part one of procedure F and the Subset Uniting Procedure in an object isolation scheme is rather high, especially when the object isolation scheme is viewed as part of a procedure to identify objects in a scene, as discussed in Chapter I. Even when using a computer as powerful as the CDC 6500, only about 20 scenes per central-processor hour can be processed (assuming the rate of one scene every three minutes is valid). Two attempts at obtaining

2	3	2	1	1	6	5	2	1	1
4	4	4	3	2	9	4	2	3	2
5	6	5	9	9	9	9	6	2	1
6	7	9	9	9	7	9	9	9	4
7	8	9	9	7	4	6	9	9	6
7	8	9	9	6	5	7	9	9	5
6	7	8	9	9	7	9	9	6	4
7	8	7	9	9	9	9	9	7	3
9	9	8	7	7	9	8	7	4	2
9	9	8	7	6	7	4	3	3	2

1	1	1	2	2	3	2	2	1	1
2	3	4	4	4	5	4	4	3	2
5	4	4	5	6	7	8	6	4	2
8	6	2	5	7	9	9	8	7	2
9	9	7	5	6	8	9	9	5	1
9	9	9	7	8	9	9	9	2	1
9	9	9	9	9	9	9	4	3	1
9	9	7	9	9	9	4	3	5	3
9	9	6	9	9	9	5	2	6	2
9	9	9	9	9	9	2	1	1	1

7	7	7	7	7	7	6	6	6	7
7	8	8	8	8	7	6	6	7	7
7	8	9	9	8	7	7	6	7	7
7	8	9	9	8	7	7	6	6	6
7	8	8	8	8	7	7	7	7	6
7	7	7	7	7	7	8	8	7	6
6	6	6	6	6	7	8	8	7	6
6	6	6	6	6	7	7	7	7	6
8	7	7	6	6	6	6	6	6	6
8	8	7	7	6	6	6	6	6	5

1	2	3	4	5	6	7	8	9	1
2	3	4	5	6	7	8	9	1	2
3	4	5	6	7	8	9	1	2	3
4	5	6	7	8	9	1	2	3	4
5	6	7	8	9	1	2	3	4	5
6	7	8	9	1	2	3	4	5	6
7	8	9	1	2	3	4	5	6	7
8	9	1	2	3	4	5	6	7	8
9	1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9	1

Figure 17. Examples of Intensity Functions

Figure 18. Mite Scene.

a faster method for object isolation in a scene sampled on a rectangular grid are discussed in Chapter III. Each of these algorithms has serious drawbacks. An algorithm which effectively overcomes the deficiency of large computational requirements and still exhibits the properties required for object isolation will be discussed in Chapter IV.

CHAPTER III

TWO APPROACHES TO THE UNIMODAL SUBSET SEPARATION PROBLEM FOR USE ON AN INTEGER GRID IN TWO DIMENSIONS

3.1 Introduction

Part one of procedure F with the Subset Uniting Procedure provides the type of separation required for object isolation. As discussed at the end of the last chapter, this method of separating unimodal subsets defined on an integer grid is not practical for object isolation problems because of the method's computational requirements. Two approaches to the problem of obtaining an algorithm which accomplishes the same results but requires less computation are presented in this chapter. Both approaches make use of the known location and ordering of points in a grid.

3.2 The Diamond-Function Procedure

The first approach is to replace part one of procedure F by another procedure which utilizes the information contained in the location and order of points on a grid. It was noted that on an integer grid, the distance from the mode of a symmetric subset to the next point which can be added to that subset is a function of the number of points already in the symmetric subset, as illustrated in Figure 19 and tabulated in Table 1. Furthermore, note that if the Manhattan metric is used, this

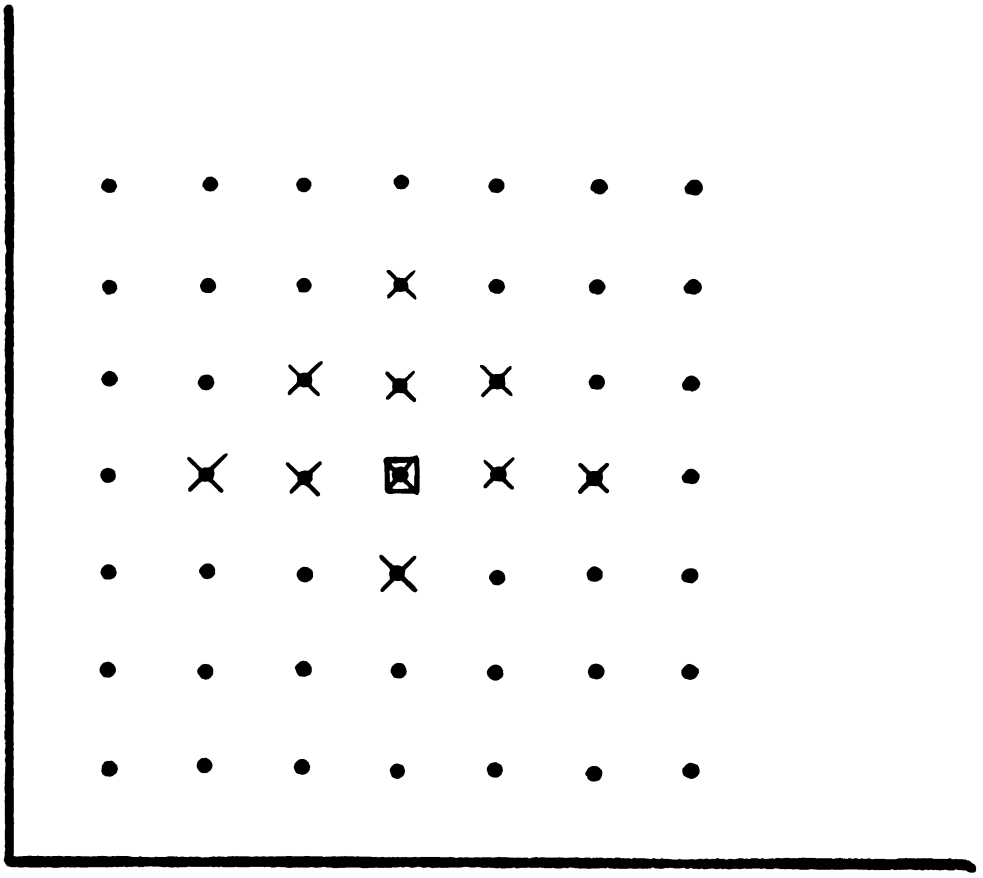


Figure 19. Symmetric Subsets on an Integer Grid

There are 10 points in the x-ed symmetric set. The next point which can be added is at distance 2 (using the Manhattan metric). There are three such points. This distance may be determined from the inequality

$2d(d + 1) \geq n > 2d(d - 1)$.

$2d(d + 1) \geq 10 > 2d(d - 1)$ implies $d = 2$, giving $12 \geq 10 > 4$.

Table 1. Distance to the Next Point Which Can be Added to a Symmetric Fuzzy Set.

NUMBER OF POINTS IN SUBSET	EUCLIDEAN DISTANCE TO THE NEXT ADDABLE POINT	NUMBER OF POINTS IN SUBSET	MANHATTAN DISTANCE TO THE NEXT ADDABLE POINT
0- 0	0.000	0- 0	0
1- 4	1.000	1- 4	1
5- 8	1.414	5- 12	2
9- 12	2.000	13- 24	3
13- 20	2.236	25- 40	4
21- 24	2.828	41- 60	5
25- 28	3.000	61- 84	6
29- 36	3.162	85- 112	7
37- 44	3.605	113- 144	8
45- 48	4.000	145- 180	9
49- 56	4.123	181- 220	10
57- 60	4.242	221- 264	11
61- 68	4.472	265- 312	12
69- 80	5.000	313- 364	13
81- 88	5.099	365- 420	14
89- 96	5.385	421- 480	15
97-100	5.656	481- 544	16
101-108	5.830	545- 612	17
109-112	6.000	613- 684	18
113-120	6.082	685- 760	19
121-128	6.324	761- 840	20
129-136	6.403	841- 924	21
137-144	6.708	925-1012	22
145-148	7.000	1013-1104	23
149-160	7.071	1105-1200	24
161-168	7.211	1201-1300	25
169-176	7.280	1301-1404	26
177-184	7.615	1405-1512	27
185-192	7.810	1513-1624	28
193-196	8.000	1625-1740	29
197-212	8.062	1741-1860	30
213-220	8.246	1861-1984	31
221-224	8.485	1985-2112	32
225-232	8.544	2113-2244	33
233-240	8.602	2245-2380	34

functional relationship between the number of points in a subset and the distance from the mode of the subset to the next addable point is particularly simple. This relationship may be derived by noting that the number of points at distance d from the mode of a subset is 4d using the Manhattan metric, for d greater than zero and d an integer (only integer distances occur), as shown in Figure 20. The number of points at a distance not greater than d from a mode (excluding the mode itself) is then $\sum_{i=1}^{d} 4i = 2d(d + 1)$. Thus, for a subset having n points (including the mode), the distance from the mode to the next addable point is a number d such that $2d(d + 1) \geq n > 2d(d - 1)$.

This relationship may be used in part one of procedure F to determine if the point to be assigned from the A_0 list, y_q^0, is assignable to any of the existing groups by simply seeing if the distance from y_q to the mode of each subset is the same as the distance to the next addable point of the subset. It is known that $n \leq 2d(d + 1)$ because y_q has not been assigned, and it is necessary only to test if d, the distance between y_q and the mode of the subset, satisfies the inequality $n > 2d(d - 1)$. If so, y_q may be assigned to this subset; otherwise, it may not. This procedure is called the Diamond-function Procedure.

The Diamond-function Procedure eliminates the need to compute and store the lists A_1, A_2, A_3, etc., which gives the Diamond-function Procedure the property of having fixed storage requirements for a given number of data points. The problem Gitman and Levine's algorithm had with equal-distance points is also eliminated, but the problem of handling

Figure 20. Contours of Constant Distance Using the Manhattan Metric.

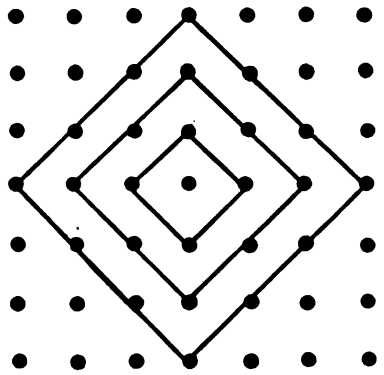
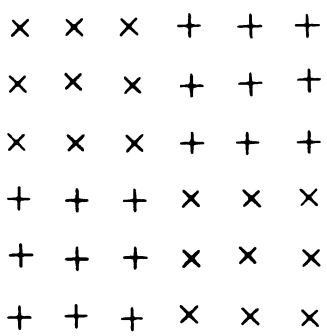


Figure 21. Crossing Connected Subsets Using an Extended Definition of Connectedness.



Both the x-ed and the +-ed sets are connected using an extended definition of connectedness.

points of equal magnitude remains. The method of processing these points is the same as in part one of procedure F. The computational considerations are also the same, except that a simple comparison of two points to see if they are identical is replaced by a distance computation, the computation of $2d(d - 1)$, and a numerical comparison to n.

Since the Diamond-function Procedure replaces part one of procedure F, either part two of procedure F followed by procedure S or the Subset Uniting Procedure may be used to complete the separation into unimodal subsets. The use of part two of procedure F and procedure S will not be discussed at length because flat regions which are local maxima may be missed by part two of procedure F. It should be noted that both part two of procedure F and procedure S can be simplified if the data points lie on a grid.

The Subset Uniting Procedure may also be simplified when the data points form a grid. The simplification is accomplished by noting that two subsets on a grid are adjacent if there is a point in one subset at a distance of one from some point in the other subset. This simplification in the Subset Uniting Procedure results in reduced computational requirements for each adjacency test, because it is no longer necessary to check all the points not in either subset for each adjacency test. The amount of computation still increases rapidly with an increasing number of symmetric subsets, however, because the required number of adjacency tests ($\frac{g(g - 1)}{2}$) remains the same. Doubling the number of subsets increases the number of adjacency tests by almost a factor of four,

an increase which leads to an unacceptable amount of computation for large numbers of subsets.

3.3 The Potential Local Maxima Procedure

The second approach taken toward the problem of reducing the amount of computation while maintaining the ability to separate a fuzzy set into unimodal subsets is described here. This procedure replaces procedure F while Procedure S is retained intact. The neighbor points, or simply neighbors, of a point are all those points at a distance of one from a particular point, using either the Euclidean or the Manhattan metric. Note that if the case of equal-magnitude points is excluded, a local maximum occurs at every point which has lower values at all four neighbor points. All the local maxima can be found by comparing each point with its neighbor points. If all the neighbor points have a lower value, a point is a local maximum. Otherwise, it is not. This procedure can replace procedure F, because both procedures do nothing more than find the local maxima. The computation requirements for the above procedure are fixed and are very low compared to the requirements for procedure F. Less than 4n magnitude comparisons need be performed to determine the local maxima, since the checking of the neighbors of each point which is not a local maximum can be terminated as soon as a higher neighbor is found. Also, the points on the edges of the grid will have fewer than four neighbor points. Doubling the number of points will roughly double the amount of computation.

Procedure S simplifies considerably in the case of a grid. Each

point is either a local maximum, which causes a new group to be generated, or it has a higher neighbor which has already been assigned to a group, due to the decreasing-magnitude order of assignment used in procedure S. Each point with a higher neighbor is assigned to the same group as was the highest of these neighbors.

This local-maxima procedure must be modified to handle the case of equal-magnitude points, because flat regions which are local maxima will be missed. The modification required is not as simple as requiring that a point is a local maxima only if none of its neighbors are higher, because that would generate many local maxima from each flat region. Many more unimodal groups than necessary would result. The interior points of each flat spot would each become a one-point unimodal region, and the property of generating one unimodal subset for each local maximum would be lost. To avoid this, it is necessary to label each point having no higher neighbor as a "potential" local maxima. Once all the potential local maxima have been found, a search for flat regions can be initiated with an arbitrary one of these potential local maximum points. Two potential local maximum points are linked if it is possible to step from one to the other through pairs of potential maximum points which are neighbors of each other. All the potential local maxima which are linked to the starting point are invalidated as local maxima and assigned to the same group as the starting point. The starting point is considered to be the mode of the group. This procedure is repeated using an unassigned local maximum point as the starting point each time until

all the potential local maxima have been assigned. Each flat region will generate only one mode in this procedure, since the potential maxima in a flat region will all be linked to each other. Procedure S will remain the same as it was for the case of no equal-magnitude points. Ties in the case of the highest-magnitude neighbor of each unassigned point may be resolved arbitrarily. The procedure described here, including procedure S, will be called the Potential Local Maxima Procedure.

Note that not all of the unimodal subsets generated by the Potential Local Maxima Procedure actually correspond to local maxima, since a flat region on the side of a "hill" (or at the bottom of a "valley") will generate potential local maximum points. One of these potential local maxima will become the mode of a group which does not contain a local maximum. The potential Local Maxima Procedure is thus not applicable to object isolation problems. A scene which is quantized into only a few intensity levels will contain "bands" of equal-intensity points. Each band will generate extra unimodal regions. This will decimate each object into small meaningless regions, instead of isolating each object as desired.

Before leaving the Potential Maxima Procedure, an interesting extension of this procedure will be discussed. It was noted that occasionally three equal-magnitude diagonally-positioned grid points with lower-valued points around them would cause the generation of three unimodal subsets, with any of the procedures previously discussed. A

question arises as to whether each of these three points generate their own unimodal set or "hill", or if they form a "ridge" as one unimodal subset. If no information is available about the continuous function from which the grid sample was made (or if there is no such function), this question would appear to be mostly a matter of individual preference and which result is desired. If the function values are sample points from a continuum, an increase in the resolution of the sample is probably indicated.

The manner in which the above question is interpreted is that the formation of symmetric subsets indirectly reflects upon the definition of connectedness in the definition of unimodal set. Gitman and Levine never directly mention anything about their concept of connectedness in a discrete point set. If the concept of "neighbor" point is extended to include the eight points closest to a particular point using the Euclidean metric, then the above would classify the three equal-magnitude diagonally-positioned points as one unimodal subset, providing that the definition of connectedness of the discrete grid is modified to accomodate such sets as being unimodal. Such a definition of connectedness is not particularly intuitively appealing because two connected subsets can cross each other, as shown in Figure 21.

If the above idea is used, another problem has been created. Suppose three points are diagonally situated as above and that all three are potential local maxima, but not of the same function value. Three cases exist: when the center point is highest, when it is lowest, and when it

is in-between the value of the end two points. In the first and last cases, one unimodal subset should be created, but in the second case, two subsets must be generated. This suggests the necessity of adding a little to the algorithm to prevent all three points from being included in one subset in the second case.

By using somewhat the same idea as was used in the Subset Uniting Procedure, the following additions will accomplish the desired result: Consider all the points having no higher neighbor (including diagonal neighbors) as potential local maxima. Start each group with the highest unassigned potential local maximum point. A link from one local maximum to another may be performed only if the value of the second is not greater than the value of the first. A slightly more general procedure may be started from an arbitrary unassigned potential local maximum point by specifying that the second point of a link may be greater than the first point only if the magnitude of the first point is the maximum magnitude which has been encountered so far in the formation of the current group.

3.4 Conclusion and Maximal Unimodal Partitions

The Diamond-function Procedure is theoretically applicable to object isolation, but in practice, the large amount of computation required when many points of equal magnitude exist (which was inherited from part one of Procedure F) makes the algorithm impractical. The Potential Local Maximum Procedure reduces the computational requirements to slightly more than linear with the number of data points

(N log (N - 1) operations are required for the initial magnitude ordering, and the rest of the procedure is linear), thus basically satisfying the linear computational requirement for an object isolation algorithm. The Potential Local Maxima procedure and the extension of it discussed above have not accomplished anything of interest to object isolation since they cause the separation of flat regions when they should not be separated. A procedure very similar to the procedure used in the extension will be the subject of Chapter IV.

Before proceeding to Chapter IV, the concept of a maximal unimodal partition will be discussed. An obvious unimodal partition for any discrete set is to consider each point as a unimodal subset. It is equally obvious that there is a "better" partition; that is, a partition with fewer subsets. By combining two of the closest points into one two-element subset, one less unimodal subset will result. In this sense, there should be a "best" unimodal partition in which the union of any two of the unimodal subsets of the partition is not unimodal. In most discrete fuzzy sets, there will be many such partitions, any of which shall be termed a "maximal unimodal partition". One unimodal subset will exist for each local maximum in the data set. The number of subsets in any maximal unimodal partition is fixed for a given data set. The property of always generating a maximal unimodal partition is a desirable property for a unimodal subset separation algorithm to possess. The lack of this property was seen to make the Potential Local Maxima Procedure unusable for object isolation.

CHAPTER IV

THE UNIMODAL TREE ALGORITHM

4.1 Introduction

Part one of procedure F, the Diamond-function Procedure and the Subset Uniting Procedure all exhibit computational requirements which increase at a rate significantly greater than a linear rate with the number of data points, as discussed previously. For this reason, these procedures are not particularly applicable to the object isolation problem. The Potential Local Maximum Procedure generates spurious separations when equal-magnitude points are present, and is therefore not applicable to object isolation. Part two of procedure F will miss most local maxima which consist of more than one equal-magnitude point, and two objects may not be separated. For this reason, any algorithm which uses part two of procedure F will not be well suited to object isolation. In this chapter, an algorithm which is applicable to object isolation will be presented. This algorithm was developed after a similarity between the Subset Uniting Procedure and the extension of the Potential Local Maxima Procedure was noted. Several definitions and a theorem which are needed to properly understand this algorithm will now be presented.

4.2 Mathematical Preliminaries

Consider a finite n-dimensional grid of points S with integer grid coordinates and a fuzzy set F in S with function value f. (S together with F may be considered to be a sample from a continuum if desired. If so, the integer grid points need not match with the coordinate system of the continuum. The entire grid may be rotated, scaled, translated, or otherwise transformed with respect to the coordinate system of the continuum. The origin of the integer grid is immaterial to the algorithm.) Let G be a graph having the set of all points in S as its vertex set and having one edge between points in each pair (a_i, b_i) satisfying $0 < d(a_i, b_i) \leq R$, where $d(a_i, b_i)$ is a metric, R is a constant, and a_i and b_i are points in S.

Definition: Two disjoint subsets S_i and S_j of S are said to be connected if there is exactly one edge of G between some element of S_i and some element of S_j. If R = 1 and a Euclidean metric is used in the definition of G, this definition corresponds to the intuitive notion of connectedness. If $R \geq \sqrt{2}$, this intuitive idea of connectedness may be violated by allowing connected subsets to cross each other in two dimensions. (See Figure 21).

Let S_{ij} be a subset of S_i, $S_{ij} = \{x \ni f(x) \geq f(x_j)\}$ for any x_j in S_i.

Definition: A unimodal fuzzy subset F_i exists in S_i if and only if the set S_{ij} is connected for all x_j in S_i. (F_i has the same function value, f, as did F).

Definition: A partition of a set S is an assignment of each element

of S to one of the k subsets S_i in a manner such that $\bigcup_{i=1}^{k} S_i = S$ and $S_i \cap S_j = \phi$ for all $i, j = 1, 2, \ldots, k$, $i \neq j$.

Definition: A partition of S is a maximal unimodal partition of S if the partition separates S into k subsets S_i, $i = 1, 2, \ldots, k$, each of which is unimodal, and if $S_i \cup S_j$ is not a unimodal subset for any $i \neq j$, $i, j = 1, 2, \ldots, k$.

An endpoint of a graph is a vertex incident to at most one edge of the graph. A tree is a connected graph containing no subgraph having no endpoints. A path is a tree with two endpoints. In a connected graph, the graphical distance between vertices v_1 and v_2, denoted by $d_g(v_1, v_2)$, is the number of edges in the shortest subpath of the graph having v_1 and v_2 as its end-points. (For a treatment of graph theory, see Behzad and Chartrand [1].)

Definition: A path P in G is called a descending path if $d_g(a, h) > d_g(b, h)$ implies that $f(a) \leq f(b)$ for all a and b which are vertices of P and where h is one of the endpoints of P, termed the high endpoint of P. The direction of the descending path is away from h.

Definition: A tree T in G is a unimodal tree if there exists a vertex h of T such that every vertex of T is a vertex of a descending path in T having h as its high endpoint.

Theorem: A unimodal fuzzy subset F_i exists in S_i if and only if G contains a unimodal tree spanning S_i. (F_i has the same function value, f, as does F).

Proof: The proof proceeds as follows: It is first shown that if

G_{ij}, the largest subgraph of G in S_{ij} ($S_{ij} = \{x \ni f(x) \geq f(x_j)\}$ for any x_j in S_i) is connected for all x_j in S_i (i.e., S_i is unimodal, from the definition), then G_i, the largest subgraph of G in S_i, has a spanning subgraph which is a unimodal tree. (By the largest subgraph is meant the subgraph having the greatest number of edges.) Second, it will be shown that if G_i is spanned by a unimodal tree, then G_{ij} is connected for all x_j in S_i, which means that S_i is a unimodal subset by definition.

1. If G_{ij} is connected for all x_j in S_i, then there is a path in each G_{ij} from every point to every other point in G_{ij}. Since every connected graph has a spanning subtree, it must be shown that at least one such spanning subtree of G_i exists which is a unimodal tree. Order the vertices in G_i by decreasing value of f. Put a spanning tree through the point(s) in G_{i1}. This can be done because of the assumption that each G_{ij} is connected (from the assumption that S_i is unimodal). This tree is unimodal since all the vertices have the same value of f. Next, add the point(s), if any, which are in G_{i2} and not in G_{i1}. (By definition all points in G_{ij} are in $G_{i\ell}$ if $j < \ell$). The tree which was put through G_{i1} is extended (edges and vertices added) to include the points in G_{i2}. This can be done because G_{i2} is connected and includes G_{i1}. The tree is still unimodal because only points of non-higher function value were added to the tree. Continue this extension of the tree to include G_{i3}, then G_{i4}, G_{i5}, and so forth, until all the points in G_i have been included in the tree. The tree is still unimodal, and it can be formed since each G_{ij} is connected and, for $j > 1$, contains $G_{i,j-1}$. Thus if S_i is unimodal in the sense of

the definition, G_i contains a unimodal tree.

2. The fact G_i is spanned by a unimodal tree means that G_i is connected. If one of the lowest-magnitude points (there may be more than one in the case of equality in function values between points) which is also an endpoint of the tree (there must be such an endpoint since the tree is unimodal) is stripped from the unimodal tree, a unimodal tree is left. Lowest-magnitude points may continue to be stripped from the tree one by one until there are no points left in the tree. At some time during this process, it will have been demonstrated that each G_{ij} is a unimodal tree, and is therefore connected. Therefore, S_i is unimodal by definition.

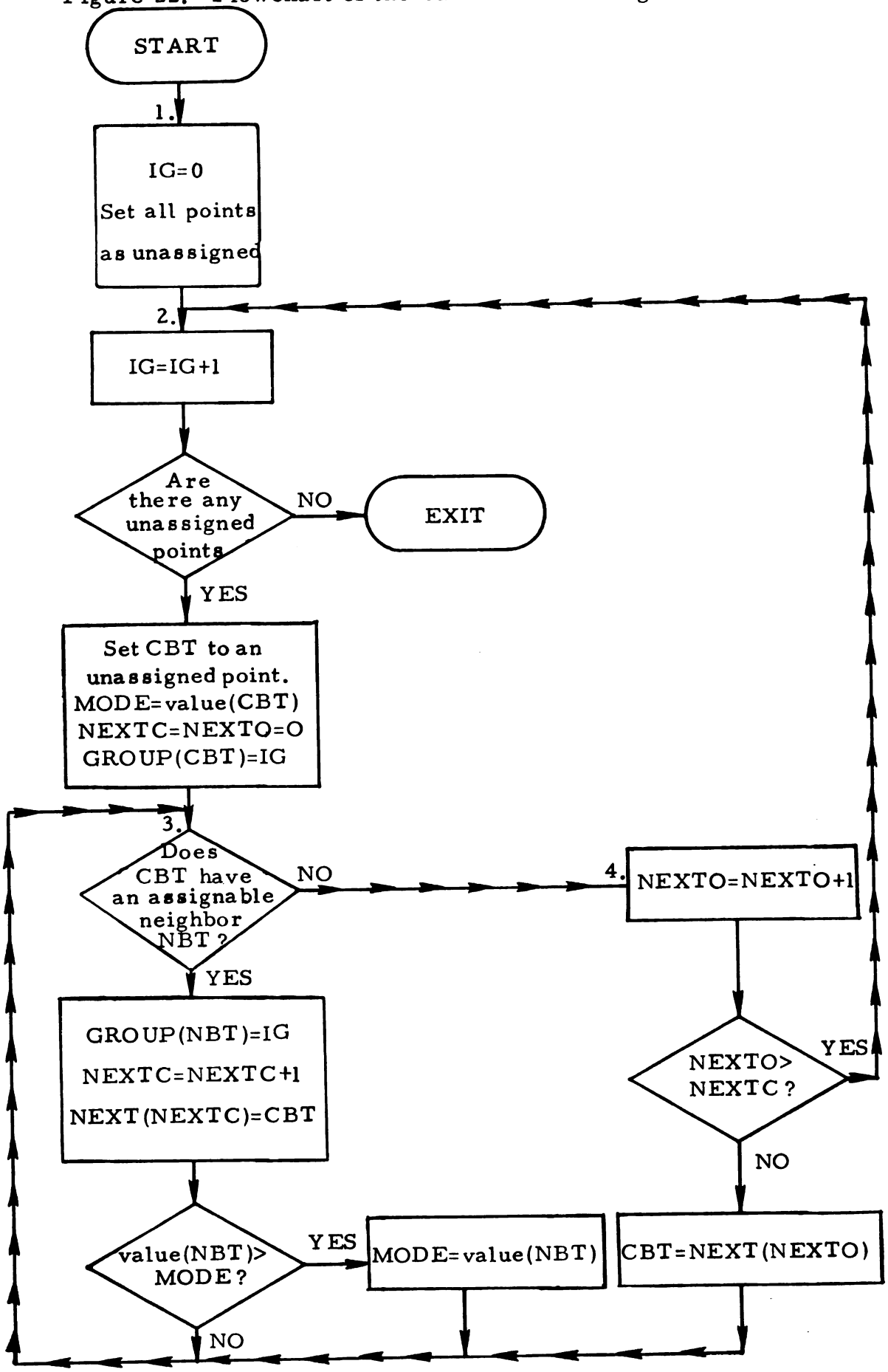
Q. E. D.

4.3 The Unimodal-Tree Algorithm

An algorithm which generates unimodal trees on an integer grid will now be described. For each subset the algorithm starts with an arbitrary point which has not been assigned to a previously-formed group and determines the largest possible unimodal tree which can be put through the currently unassigned points. The first subset will have as many points as possible, the second will be as large as it can be without using points from the first subset, and so forth. The procedure is summarized below, and a flow chart is shown in Figure 22.

Several arrays, pointers, a flag array, and a variable are needed. It is assumed that the function value of each point is in one array.

Figure 22. Flowchart of the Unimodal Tree Algorithm.



Another array, called NEXT, is needed to record branching points so that other branches from each point not an endpoint of the unimodal tree being formed may be found later. The array NEXT functions as a stack. A first-in first-out stack was used here, but the order of removing points which need to be checked for further branching is immaterial. Pointer NEXTC points to the last entry in NEXT and pointer NEXTO points to the last element removed from NEXT. Another array GROUP is needed to hold the group number of each point as it is assigned. An array of flags is necessary to tell which points have been assigned. Two points which are linked by exactly one edge of G are said to be neighbor points. It is assumed that a means exists for determining all of the neighbors of a given point. A group counter IG is needed to count the number of groups formed. A variable MODE which has the same value as the highest point found in each group as it is being formed is required. A pointer CBT, standing for Current Branch poinT, is needed to point to the place at which the unimodal tree is currently being expanded, and another pointer NBT, for Next Branch poinT, is needed to point to an assignable neighbor of CBT.

The steps of the Unimodal Tree Algorithm follow.

1. (Initialization of the algorithm). Set IG to zero to indicate that no groups currently exist. Set the assignment flag for all points to indicate that every point is currently unassigned. Continue with step 2.

2. (Initialization of a new group). Set IG to IG + 1 in order to start the next group. If possible, find an arbitrary unassigned point and set

CBT to point to it. (This will be the starting point for the new group). If it was not possible to find an unassigned point, go to step 5. If it was possible, set MODE to the value of CBT. (This is the highest value which has occurred in this group). Set NEXTC to zero to indicate that no points have been entered into stack NEXT. Set NEXTO to zero to indicate that no points have been removed from NEXT. Set GROUP(CBT) to IG (this assigns CBT to group IG). Continue with step 3.

3. (Linkage to an assignable neighbor). If possible, find a point NBT such that NBT is unassigned and a neighbor of CBT and, if the value of CBT is less than MODE, such that the value of NBT is not greater than the value of CBT. (These conditions on NBT are necessary to preserve unimodality in the unimodal tree, as discussed below). If this is not possible, go to 4. If it is possible, set GROUP(NBT) to IG. Set NEXTC to NEXTC+1 and set NEXT(NEXTC) to CBT. (This enters CBT into stack NEXT so that the point can be checked later for additional assignable neighbors). If the value of NBT is greater than MODE, set MODE to the value of NBT. (This maintains MODE as the highest value yet found in the current group). Set CBT to NBT and repeat 3. (This accomplishes a linkage to NBT).

4. (An old point which was branched from in step 3 is obtained from stack NEXT and rechecked). Set NEXTO to NEXTO+1. If NEXTO > NEXTC go to 2. (This gets the next old branch point location in stack NEXT. If there is no such point left in NEXT, a new group is started in step 2). Otherwise, set CBT to NEXT(NEXTO) and go to 3.

5. (Exit). The data is now partitioned into a maximal unimodal partition.

To verify that this algorithm generates a unimodal partition, an inductive argument is presented which shows that each group must be a unimodal tree. If each group is a unimodal tree, it is then a unimodal subset, using the theorem. The inductive argument proceeds by stating that the starting point of each group is a unimodal tree and then showing that whenever step 3 of the procedure is applied to one of the endpoints of a unimodal tree (as CBT), a unimodal tree will result. Given a unimodal tree, each endpoint of the tree has either the highest magnitude in the tree or it does not. If the particular endpoint in question, CBT, has the highest magnitude in the tree, step 3 will allow any unassigned neighbor point of CBT to extend the tree. Regardless of whether this point is higher, the same, or lower in magnitude, the resulting tree is unimodal. If it is higher, all the descending paths from point H ((one of) the largest magnitude point(s) of the tree) to all the other points of the tree may be extended to include the new point as a highest point of the extended tree, which means the tree is unimodal. If the point is equal to or lower than the mode of the original tree in value, the descending path from H to CBT is extended by one edge to include the new point. Thus all points in the extended tree are included in descending paths each having (one of) the largest magnitude point(s) of the extended tree as its high endpoint, and the extended tree is therefore unimodal. In the case where CBT does not have the largest magnitude in the original

tree, step 3 allows extension of the tree only to points having value less than or equal to the value of CBT. Since CBT resides on the low endpoint of a descending path from point H, this descending path may be extended to include the new point. Thus the application of step 3 will always generate a unimodal tree, providing the tree given to step 3 is a unimodal tree. Since the starting point by itself is a unimodal tree, as given to step 3 by step 2, the algorithm will always generate unimodal trees, and consequently (by the theorem) unimodal subsets.

To verify that this algorithm generates a maximal unimodal partition, it is necessary to show that all the subsets are unimodal, which has been done, and that the union of any two subsets generated is not unimodal. Note that the algorithm always follows descending paths as far as they can possibly be extended, in either the ascending or descending direction. This is ensured by saving each point along each descending path as the path is extended, and rechecking the points later for further branches. If further branches are found, the point is again saved and later restored for further checking. Therefore, the particular unimodal tree being formed is extended as far as possible. Expansion of a unimodal tree stops only when every unassigned neighbor of every endpoint of the tree violates the condition that the addition of that neighbor would maintain a descending path from the mode of the tree to that neighbor. Suppose S_i and S_j are two unimodal subsets generated by the algorithm and that $S_i \cup S_j$ is also unimodal. Either S_i or S_j must be generated first. Suppose S_i was generated first. Because $S_i \cup S_j$ is unimodal, there must

be endpoints of the unimodal tree generated in S_i which are neighbors of some points in S_j. Furthermore, a descending path exists between the mode of S_i and S_j, in the direction of the higher to the lower (or in either direction if the modes are equal). The part of this path in S_i may be replaced by a path between the mode and the boundary which was generated by the algorithm, since the direction of a descending path is defined by the relative magnitudes of its endpoints. At the crossover between the two subsets, the path generated by the algorithm has a continuation into S_j. The algorithm will then assign a point of S_j to S_i. But the point is in S_j, so the assumption that the algorithm can generate two unimodal subsets the union of of which is also unimodal has been contradicted. The algorithm, therefore, always generates a maximal unimodal partition.

4.4 Implementation of the Unimodal Tree Algorithm

The Unimodal Tree Algorithm has been implemented in FORTRAN II, in machine-independent code. The implementation described here is for R=1 in the definition of graph G, using the Euclidean metric. The starting point for each group is the highest unassigned point. After a group has been completely assigned, there is no further use for the magnitude of the points in that group; therefore, the number of the group may be stored in place of the magnitude value. This technique enables the GROUP array to be the same as the array which holds the function values, and reduces the storage requirements by N, if there are N data points. The magnitude value of each point cannot be discarded until the point is observed not to have any assignable neighbors. Thus

the assignment flag for each point is turned on immediately when the point is assigned, but the group number is not inserted until the point is found to have no assignable neighbors; that is, the setting of GROUP(CBT) to IG is removed from step 3 and inserted as the first thing in step 4. This may be done since each point is rechecked through being entered and re-entered in the stack NEXT until all assignable neighbors are assigned. The assignment flag procedure was implemented by requiring that the data be positive. All the data points were changed in sign before execution of the algorithm. A negative value for a point indicates the point has not been assigned. When the assignment flag is set, the data value is changed back to its original positive value. Then when the necessarily positive group number is inserted into the data value location, the assignment flag, which is the sign of the data value, remains positive, indicating assignment of that point.

The storage requirement of the Unimodal Tree Algorithm as implemented is 2N storage locations for N grid points, plus 3*NDIM locations for an NDIM-dimensional grid, plus the program requirements. N of these locations are needed to store the input data values and the output group numbers. N more locations are used for stack NEXT. Note that while some points may be entered more than once in stack NEXT, there is a point for each extra entry into NEXT which does not get entered into NEXT at all. This point is the last point assigned before the next entry is taken from the stack. NDIM locations are required to store the number of grid increments in each dimension, and NDIM more locations

are required apiece for two coordinate vectors, one for CBT and one for NBT.

The number of computations involved in this algorithm varies slightly from scene to scene because of the variable length of searches for an assignable neighbor for various points. All the points are searched at least once and most of them twice. Each time a point is entered into the NEXT table, it will be searched again in the future. The total number of searches performed is 2N-g where there are N points input and g groups formed. This may be verified by noting that each duplicate entry into stack NEXT is offset by a point which is not entered at all. In addition, the point which was assigned just before the first point was taken from the NEXT array is not entered into the NEXT array and is not checked again. This happens g times so 2N-g checks are performed. In the current implementation, all the neighbors of each point are checked until an assignable neighbor is found, so the tests are of different lengths depending upon which, if any, of the neighbors are assignable. If additional storage is available, the number of the next neighbor to be checked for each point in stack NEXT could be recorded. Then each neighbor position of each point would be checked exactly once and the number of computations would be fixed. The added complexity and storage requirements to do this were prejudged as not justifiable, so no implementation of this scheme was attempted. A savings in computation time may be obtained at the expense of additional storage requirements. Note that only a few bits are required to store the number of the next neighbor

(4 bits will accomodate up to 8 dimensions). A few bits of each data word may be used to store this information, if a degradation in the resolution of the data can be tolerated. If a large number of bits per word are available, the data value, the assignment flag, an entry in the NEXT stack, and the number of the next neighbor to be checked may all be stored in one word, reducing the core requirements to N locations.

The principal feature of the Unimodal Tree Algorithm is the approximately linear amount of computation and storage required by the algorithm. In addition, the Unimodal Tree Algorithm is not adversely affected by points of equal magnitude. The Unimodal Tree Algorithm generates a maximal unimodal partition, which provides a great advantage over the Potential Local Maxima Procedure followed by procedure S. The Unimodal Tree Algorithm thus appears to fulfill the requirements needed for an object isolation scheme. Application of the Unimodal Tree Algorithm to object isolation and to clustering of points will be discussed in the next chapter.

CHAPTER V

APPLICATIONS OF THE UNIMODAL TREE ALGORITHM

5.1 Object Isolation

Two applications of the Unimodal Tree Algorithm are presented in this chapter. The first application is object isolation and the second is clustering of points in a space. Object isolation means defining "regions of interest" in a two-dimensional image. These "regions" of interest" could correspond to actual objects in a sampled scene or to portions of the objects. Objects in a scene can be separated by this algorithm only if each object corresponds to a unimodal set. Most images contain far more data points than are needed, and they also contain noise and many fluctuations in intensity which are meaningless for object isolation. (See Figure 23.) Direct application of a sampled image to the Unimodal Tree Algorithm would result in a myriad of small meaningless unimodal subsets being generated. A procedure to "enhance" the image for object isolation is thus necessary. Such a procedure is termed preprocessing of the image. The intent of preprocessing an image before object isolation is to reduce the number of data points so that the objects can still be distinguished but the details of each object, which are meaningless in object isolation, are lost. If the resolution cannot be reduced so that the objects are dis-

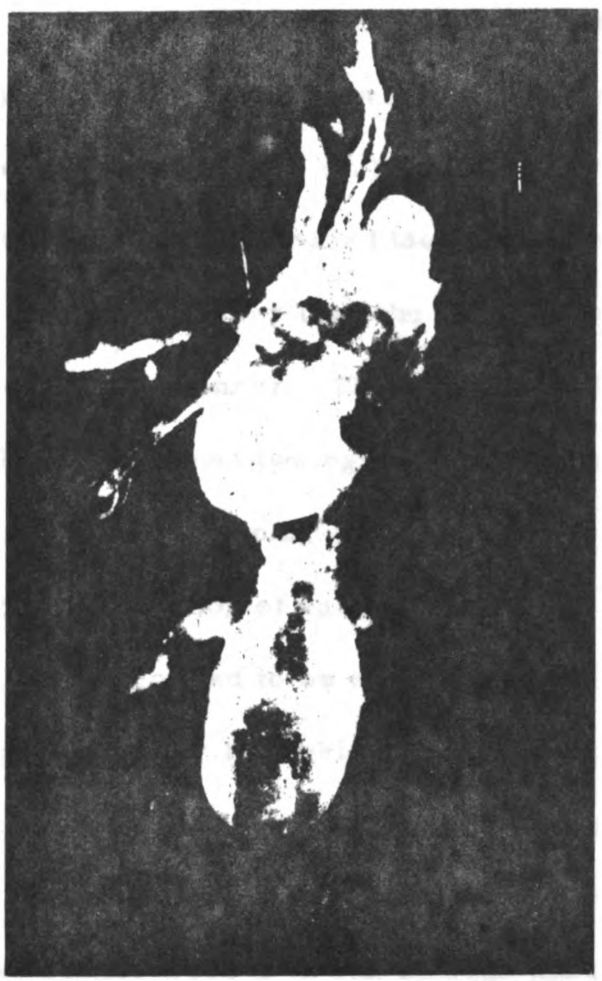


Figure 23. Mite Scene.

tinguishable but object detail is lost, further "smoothing" of the reduced image may be required.

5.2 Object Isolation Examples

The scenes which were scanned and processed for this thesis were scanned on a 100 x 100 grid, giving 10,000 data points per scene. The scanned scenes were reproduced on a line printer with eight distinct brightness levels. (See Figure 24.) All scenes were then reduced to a 25 x 25 grid by averaging square blocks of sixteen points each. (See Figure 25.) Two photos other than the mite photo of Figure 23 were also processed in this manner. They are shown in Figure 2 and Figure 26. Figure 2 shows two eggs touching each other and Figure 26 shows three eggs and three ping-pong balls, none touching each other.

The original resolution of each of the 10,000 data points in one of these scenes was estimated to be eight bits (256 levels). This was reduced to approximately 16 levels of brightness. This reduction of the number of quantization levels of the image resulted in some "smoothing" of the undesirable fluctuations in the scene.

The three scenes were then run through the Unimodal Tree Algorithm. If the dark background level was suppressed, the two eggs of Figure 2 and the six objects of Figure 26 were isolated. Two and six unimodal groups were generated, respectively, and each group corresponded to an object. The mite scene, which still retained some detail of the mites, was separated into 21 unimodal regions. (This is the

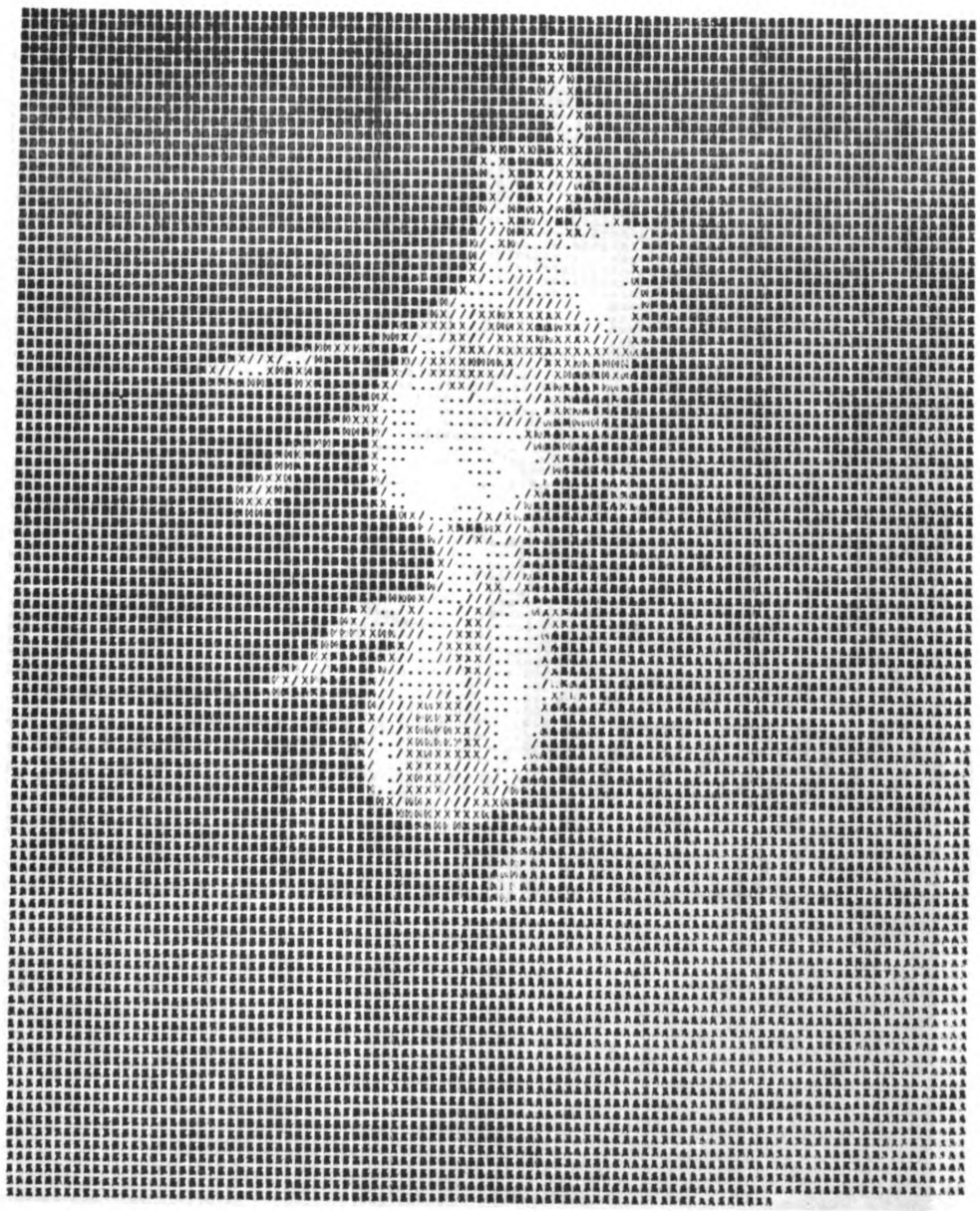


Figure 24. Scanned Mite Scene.

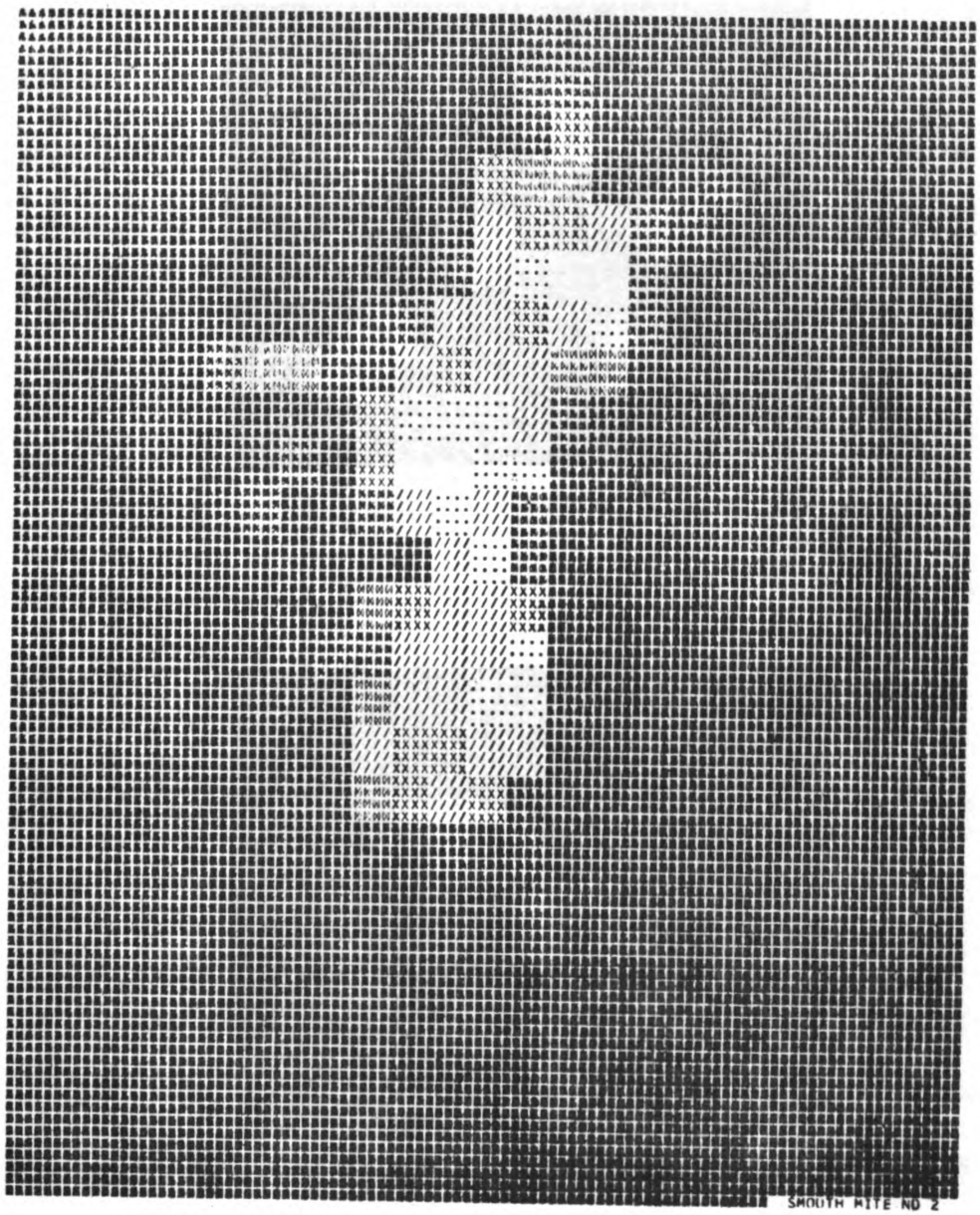


Figure 25. 25 x 25 Grid of the Mite Scene

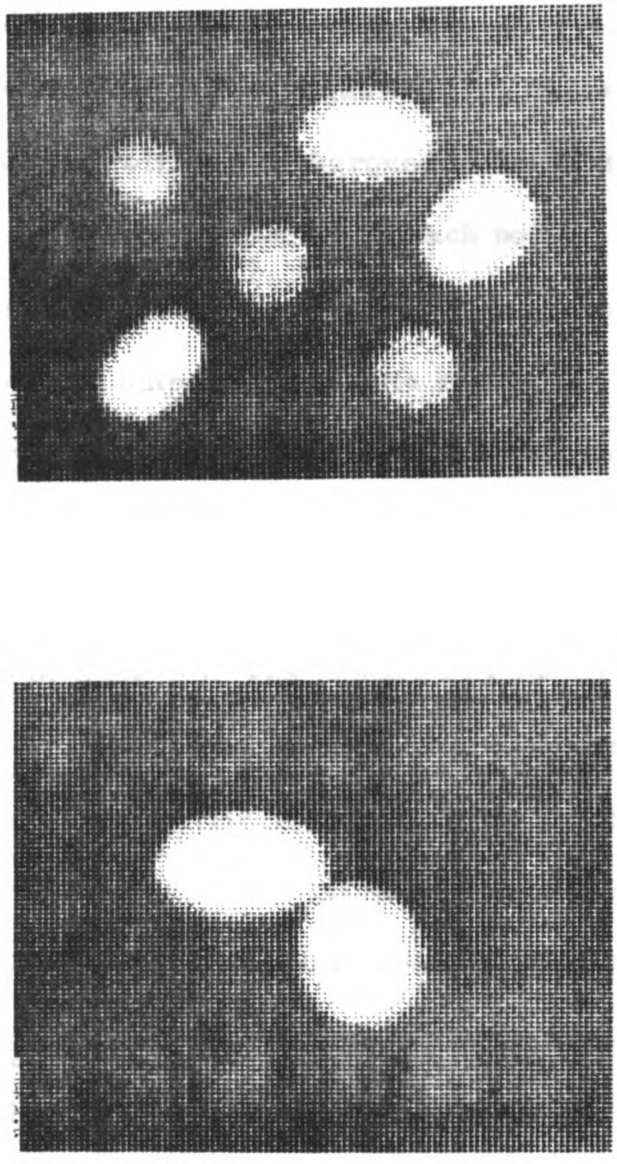


Figure 26. Ellipsoid and Sphere Scenes.

mite scene upon which part one of procedure F and the Subset Uniting Procedure were run as described in Chapter 2. Some of the background was cut off to provide less than 500 points for that run.) The output from these runs is presented in Figures 27, 28, 29, 30, 31, and 32. Figure 27 shows the input to the algorithm for the touching eggs, and Figure 28 the output, with the background set to zero. The integers in Figure 28 indicate the group number of each point. Figures 29 and 30 show the input and output for the six-object scene, and Figures 31 and 32 show the input and output for the mite scene.

Additional "smoothing" was performed on the mite picture. Each point of the scene was replaced with a weighted average of itself and its neighbor points; that is, I_{ij}, the intensity of the ij^{th} grid point (not on the boundary of the grid), was replaced by $[W_1 * I_{ij} + W_2 * (I_{i-1,j} + I_{i+1,j} + I_{i,j-1} + I_{i,j+1})]/(W_1 + 4W_2)$, where W_1 and W_2 are arbitrary weighting constants (a similar average is used for the boundary points). This process was carried out N_{av} times. "Smoothing" or averaging action is displayed by this procedure when W_1 and W_2 are both positive. For $W_1 = 4$, $W_2 = 1$, and $N_{av} = 3$ (these numbers are intelligent guesses updated by trial and error), three subsets were generated instead of 21. (See Figure 33.) The three groups corresponded to the smaller of the two mites and to the two ends of the larger mite. The larger mite was split into two groups because the ends of the mite were lighter than its mid-section.

INPUT DATA

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	2	3	3	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	3	9	12	13	13	10	2	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	12	15	15	15	15	15	11	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	7	14	15	15	16	15	15	12	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	3	5	14	15	15	15	15	14	6	1	1	1	1	1	1	1
1	1	1	1	1	1	1	3	12	14	13	9	11	13	12	10	5	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	11	15	15	15	11	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	2	14	15	15	15	14	2	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	2	14	15	15	15	14	2	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	13	15	15	15	14	2	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	11	15	15	15	12	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	3	10	13	11	4	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 27. Two Touching Ellipsoids After Averaging.

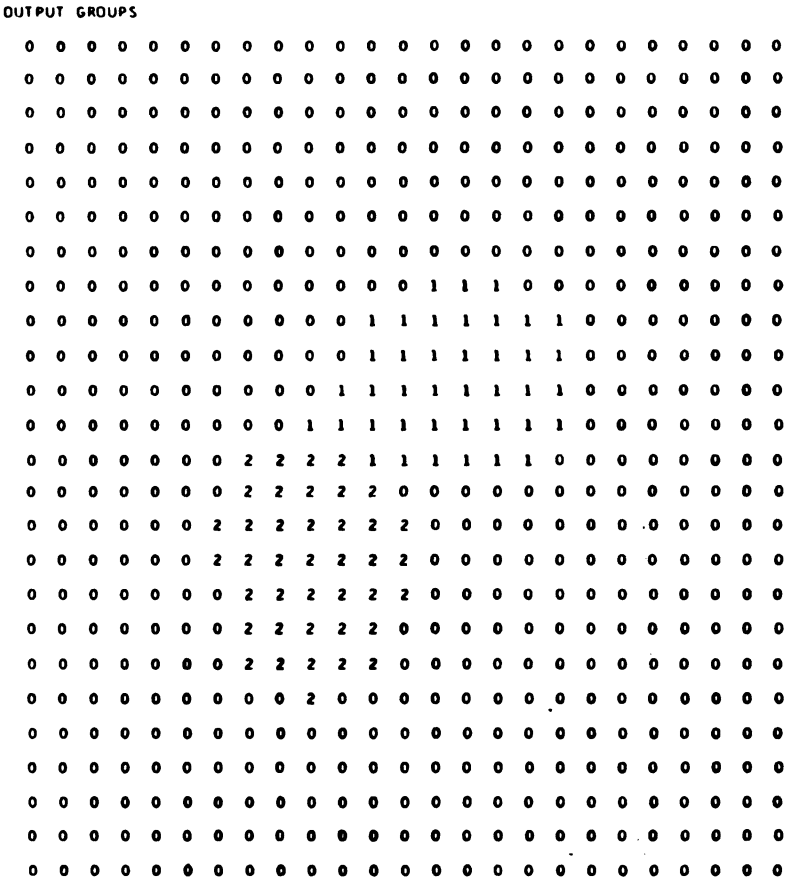


Figure 28. Two Touching Ellipsoids as Separated by the Unimodal Tree Algorithm.

INPUT DATA

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	8	13	12	7	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	2	14	15	15	15	8	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	2	13	15	15	15	14	2	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	6	13	14	14	14	2	1	1	2	6	6	2	1	1	1	1
1	1	1	1	1	6	11	8	1	1	3	7	8	5	1	1	1	9	11	12	7	1	1	1
1	1	1	1	5	14	15	15	6	1	1	1	1	1	1	1	1	8	11	10	6	1	1	1
1	1	1	1	11	15	15	15	10	1	1	1	1	1	1	1	1	3	6	5	2	1	1	1
1	1	1	1	11	15	16	15	10	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	6	14	15	13	3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	5	7	4	1	1	1	1	5	6	2	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	7	11	12	9	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	6	11	11	9	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	2	5	6	3	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3	2	1	1	1	1	1
1	1	1	1	1	1	1	2	3	1	1	1	1	1	1	2	12	14	13	8	2	1	1	1
1	1	1	1	1	1	3	9	10	8	1	1	1	1	1	4	14	15	15	15	11	1	1	1
1	1	1	1	1	1	6	10	12	9	2	1	1	1	1	1	10	14	15	15	14	3	1	1
1	1	1	1	1	1	3	7	8	6	1	1	1	1	1	1	2	8	13	14	12	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3	4	2	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 29. Six-object Scene After Averaging.

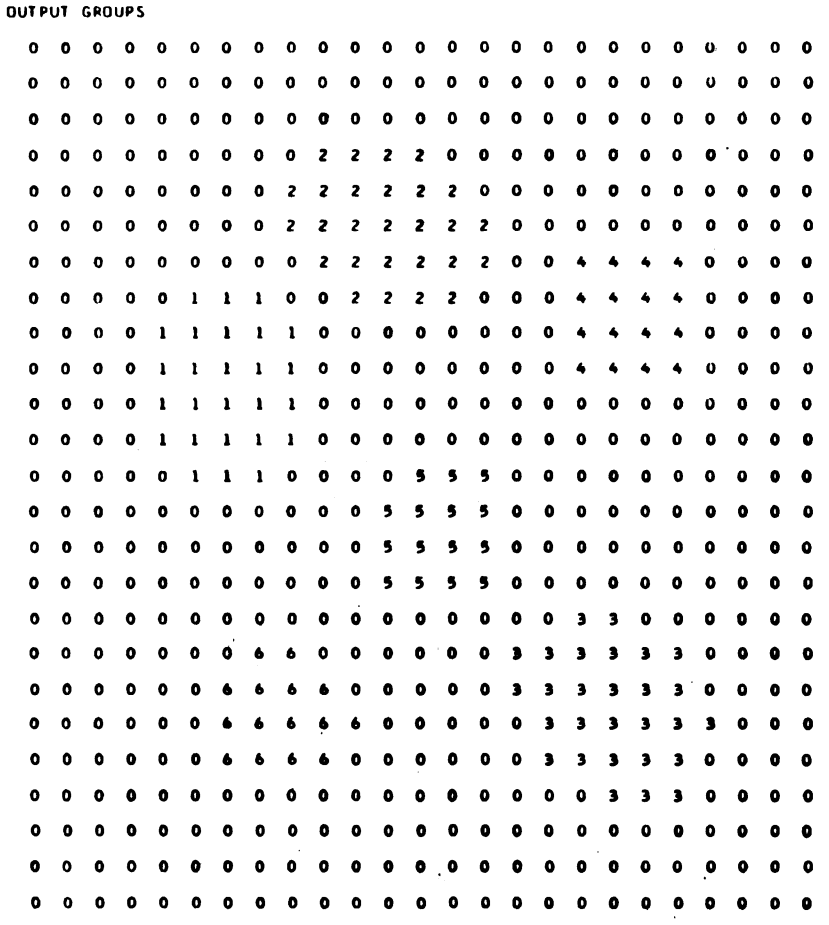


Figure 30. Six Objects Isolated by the Unimodal Tree Algorithm.

INPUT DATA

2	2	2	2	2	2	2	2	2	2	2	2	2	5	2	2	2	2	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	11	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	3	5	5	1	1	1	1	2	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	2	9	10	3	1	1	2	2	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	5	8	12	11	9	2	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	10	12	13	15	15	3	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	3	8	12	9	8	12	14	2	1	1	1	1	1	1	1	1
1	1	1	1	3	7	5	6	2	6	11	9	11	10	6	7	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	6	13	15	13	13	8	5	2	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	5	3	1	14	16	14	15	9	2	2	1	1	1	1	1	1	1	1	1
1	1	1	1	1	5	4	1	1	7	14	12	12	2	1	3	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	4	13	12	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	4	8	11	11	14	6	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	6	4	8	13	9	14	9	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	4	3	1	12	10	11	14	10	2	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	13	7	10	14	3	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	3	1	10	9	11	8	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	2	1	1	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	7	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 31. Mite Scene After Averaging

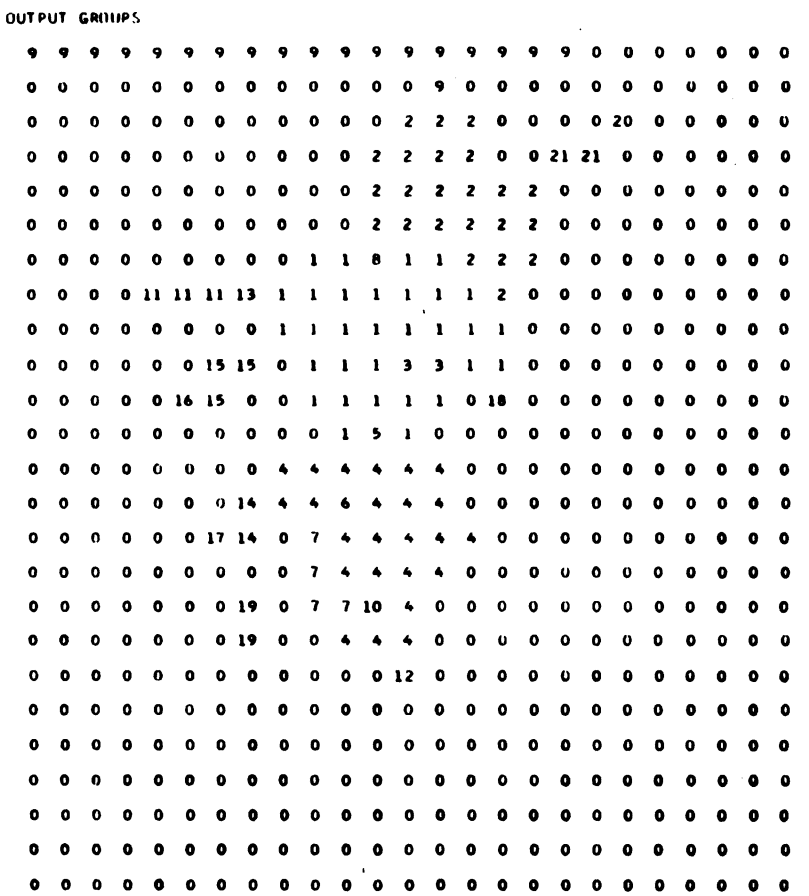


Figure 32. Mite Scene as Separated by the Unimodal Tree Algorithm.

SMOOTHED DATA   IW1 = 4   IW2 = 1   NAV = 3

1	1	1	1	1	1	1	1	1	1	1	1	2	3	2	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	2	4	2	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	3	4	3	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	3	5	6	4	2	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	4	7	9	8	6	2	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	3	6	9	10	11	9	4	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	3	6	8	9	9	9	8	3	1	1	1	1	1	1	1	1
1	1	1	1	1	2	2	2	3	6	8	10	9	8	6	4	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	2	2	4	9	12	11	10	7	4	2	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	2	2	3	9	12	12	11	6	3	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	2	6	9	10	9	4	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	4	7	9	9	4	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	2	3	6	8	10	10	5	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	2	4	7	9	10	10	6	2	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	2	3	7	9	9	10	6	2	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	3	7	8	8	8	4	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	2	5	6	7	5	2	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	2	2	3	3	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 33. Enhanced Mite Scene Input and Output.
A. "Smoothed" Mite Scene.

OUTPUT GROUPS

0	0	0	0	0	0	0	0	0	0	0	0	1	2	2	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	2	2	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	2	2	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	1	2	2	2	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	1	2	2	2	2	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1	1	1	2	2	2	2	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	3	3	3	3	3	3	3	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	3	3	3	3	3	3	3	3	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	3	3	3	3	3	3	3	3	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	3	3	3	3	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 33. Enhanced Mite Scene Input and Output.
B. Separation of "smoothed" mite scene by the Unimodal Tree Algorithm.

The above three examples show that the Unimodal Tree Algorithm is capable of performing object isolation providing that proper pre-processing of the image is performed. There is certainly much more to be done in the application of the Unimodal Tree Algorithm to object isolation, but pursuing the matter further is not the intent of this thesis.

5.3 Clustering

The second application suggested here is the clustering of points in a space. The primary purpose of a clustering algorithm is to label points which are "similar" according to some measure with the same label, and to give points which are "not similar" different labelings. (See Zahn [10] and Gitman and Levine [5].) The only input information to a clustering algorithm is normally the location of data points in the space and a few parameters which relate to the general form of the expected clustering results. Examples of such parameters are the number of clusters to be found and parameters relating to the resolution of the clusters. The clustering scheme which is presented here will use an approximate "density function" of the data on a grid covering the range of the data. Conceptually, a density function should have a "high" value where data points are clustered tightly together and a "low" value where data points are sparsely spaced. In this sense, many functions qualify as density functions. A density function which is straightforward to compute has been selected for use in the examples which follow. Given a grid lying in the range of the data, the density

function is formed by counting the number of data points closer to a particular grid point than to any other, and assigning this count as the function value of the grid point. This is done for all the grid points. (Data points lying outside the boundaries of the grid should be excluded.)

It is necessary to determine the region of the data space over which the grid should extend. The grid selected will lie in a rectangular hyper-parallelepiped. Two suggestions for selecting the size of the hyper-parallelepiped are to include all the data points in the smallest possible rectangular hyper-parallelepiped which has sides parallel to the axes of the data space, or to set the boundaries of the hyper-parallelepiped to the centroid of the data set plus and minus several standard deviations in each dimension of the data set. An advantage of the second scheme is that single points which lie far away from the other data will be excluded. If these "outliers" are included, as in the first method, a degradation of the resolution of the density function in the "good" region of the data set will result, if the number of grid points is held constant. More grid points could increase the resolution back to the original level, but a lot of storage and computation would be wasted on a few "outliers" which probably represent erroneous points in the data set. On the other hand, if all the data is "good," setting the boundaries to the centroid plus and minus, say, 3 standard deviations in each dimension may set the boundaries further away from the data set than they need be, which would also reduce the resolution

which may be obtained with a given number of grid points. A composite suggestion is to select the boundary closest to the centroid of the data set in each case. Note that if the computer on which this clustering scheme is to be implemented has floating-point software in lieu of hardware (that is, subroutines to perform floating-point arithmetic instead of computer instructions), finding the boundaries using the standard-deviation method may require more computation time than the actual separation of clusters using the Unimodal Tree Algorithm.

Due to the random nature of most data sets used in clustering, "smoothing" of the density function (which was formed by counting the number of points closer to each grid point than to any other) is necessary. The smoothing technique which was used for the preprocessing in object isolation was applied to the density function. This smoothed density function was run through the Unimodal Tree Algorithm to determine all the density "hills." Data points were then labeled by assigning each data point to the cluster corresponding to the unimodal subset to which the nearest grid point belonged. Note that the case of distance ties causes no trouble either in defining the density function or in assignment of points to clusters. The choice is arbitrary in each case.

Some advantages to note about this clustering scheme are the following: 1. The clusters formed are independent of cluster shape. (See Zahn [10] for a discussion of cluster shapes.) 2. Computation and storage requirements are both approximately proportional to the number of grid points, except for the formation of the density function and the

labeling of the data points, where the computation requirements are proportional to the number of data points. The data need not be in high-speed storage all at once if mass storage facilities are available. Only one data point need ever be in core at a time, so that the total core-storage requirements may possibly be less than the number of data points. 3. Clusterings are independent of scaling changes in the data. 4. Clusterings are independent of the input data order. 5. An increase in the number of data points will help this scheme to cluster, because if the data is thought of as being random samples from a governing distribution, the density function will approach the "true" density as the number of data points grows large.

Some disadvantage of this scheme are: 1. "Good" clustering results are obtained only when using "sufficient" resolution in defining the grid and "enough" smoothing of the density function. 2. The number of grid points needed for a constant resolution per feature increases astronomically with the number of dimensions. A 10-step resolution for each feature requires 100 grid points for two dimensions, 10,000 for four dimensions, and 10^{10} for ten features. The practical limit to the number of features would thus appear to be four or five features. A possible partial alleviation of this problem might be to take the number of grid steps in each dimension in relation to a measure of the "importance" of that dimension. 3. Results are dependent upon the number of grid points and the smoothing employed. 4. Clusters of uniform density generate small random meaningless fluctuations in the density

function which are then separated into equally meaningless clusters. A scheme to rectify this complaint might be to form a histogram of the relative frequency of occurrence of each density magnitude on the grid. Uniform clusters should generate spikes in this histogram. Each grid point having a value lying in one of the spikes may be set to the mean value of its spike. This should "flatten out" the random fluctuations of the uniform cluster, and enable the desired clustering to result.

5.4 Experiments with Clustering

Several data sets were applied to this clustering scheme. The most notable data which was run is the Fisher Iris Data [3]. This data is four-dimensional, and was run in all four dimensions. For a 5 x 5 x 5 x 5 grid, with $W_1 = 6$, $W_2 = 1$, and $N_{av} = 1$ (these numbers are intelligent guesses updated by trial and error), 15 out of 150 Iris patterns were misclassified. Several bivariate Gaussian data sets were run, some of them having severely overlapping clusters. (See Figure 34.) In all cases, the correct number of clusters were generated with the proper selection of the grid size and the constants W_1, W_2, and N_{av}. Usually, two or three runs were necessary to select good parameter values. In addition, a bivariate data set containing a Normal cluster surrounded by a crescent of points was run. (See Figure 35.) All the 200 points in this data set were labeled into two groups corresponding exactly to the original clusters. This last run

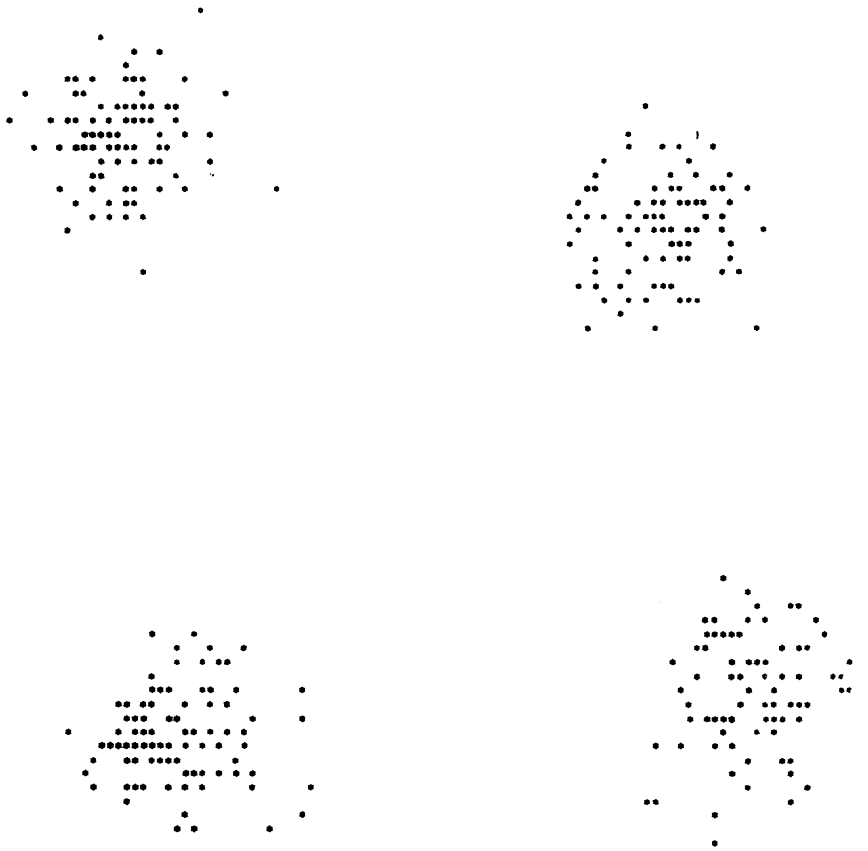


Figure 34A. Bivariate Gaussian Clusters.
Four well-separated Gaussian clusters, unseparated.

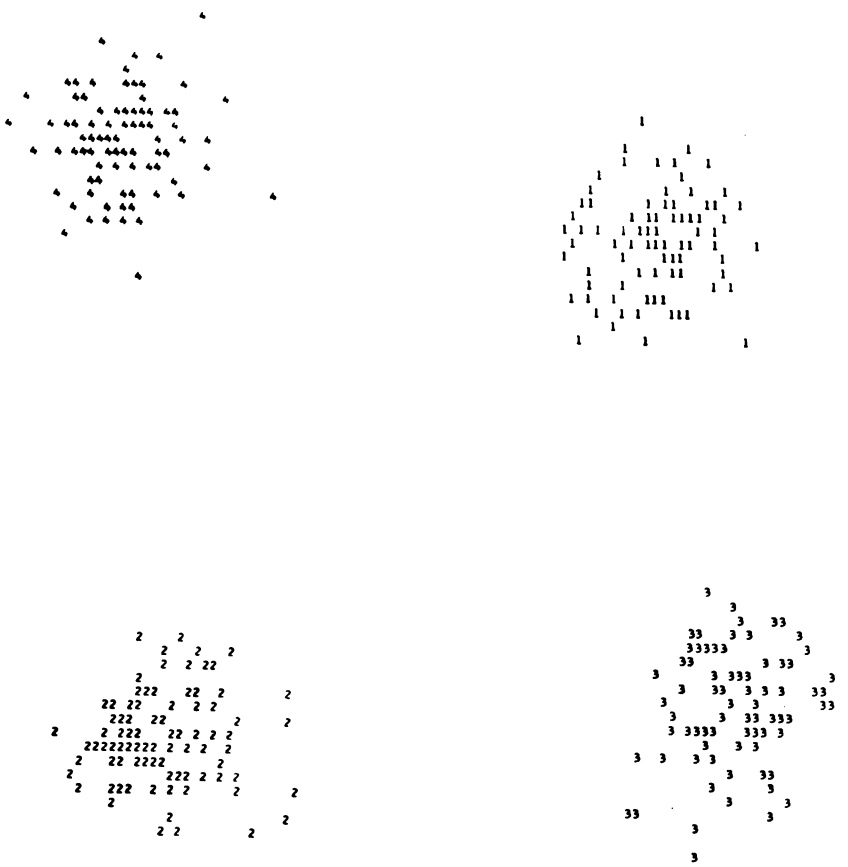


Figure 34B. Bivariate Gaussian Clusters.
Four clusters of A, as generated.

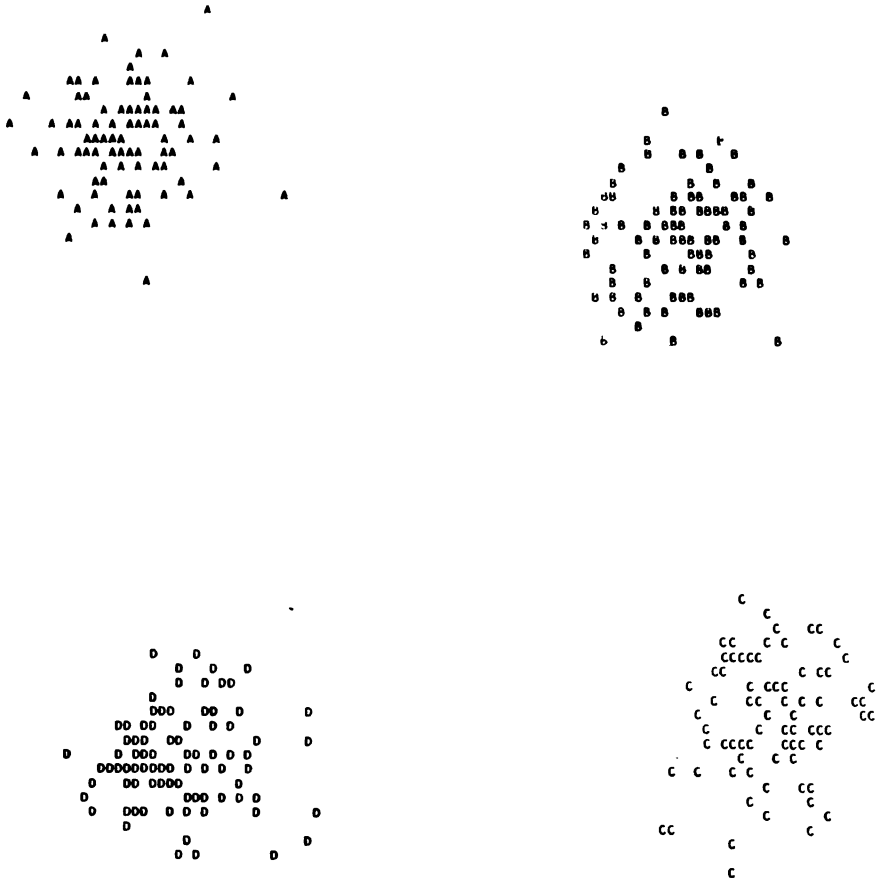


Figure 34C. Bivariate Gaussian Clusters.
Four clusters of A, as separated by the Unimodal Tree Algorithm.

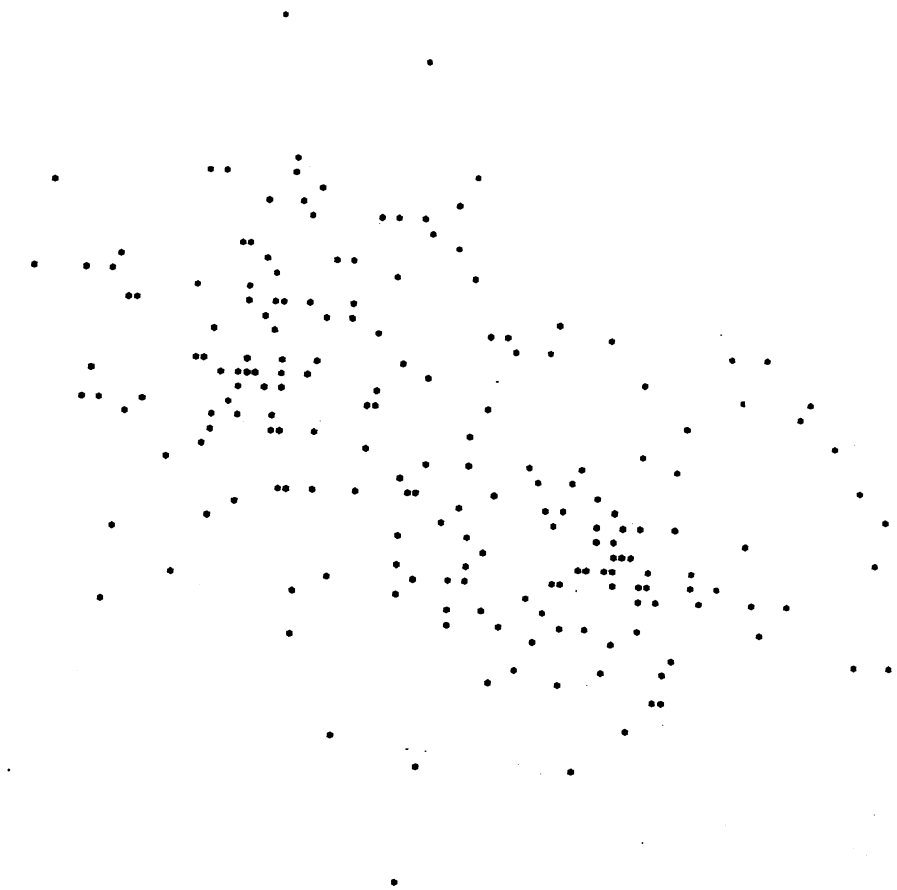


Figure 34D. Bivariate Gaussian Clusters.
Two overlapping Gaussian clusters, unseparated.

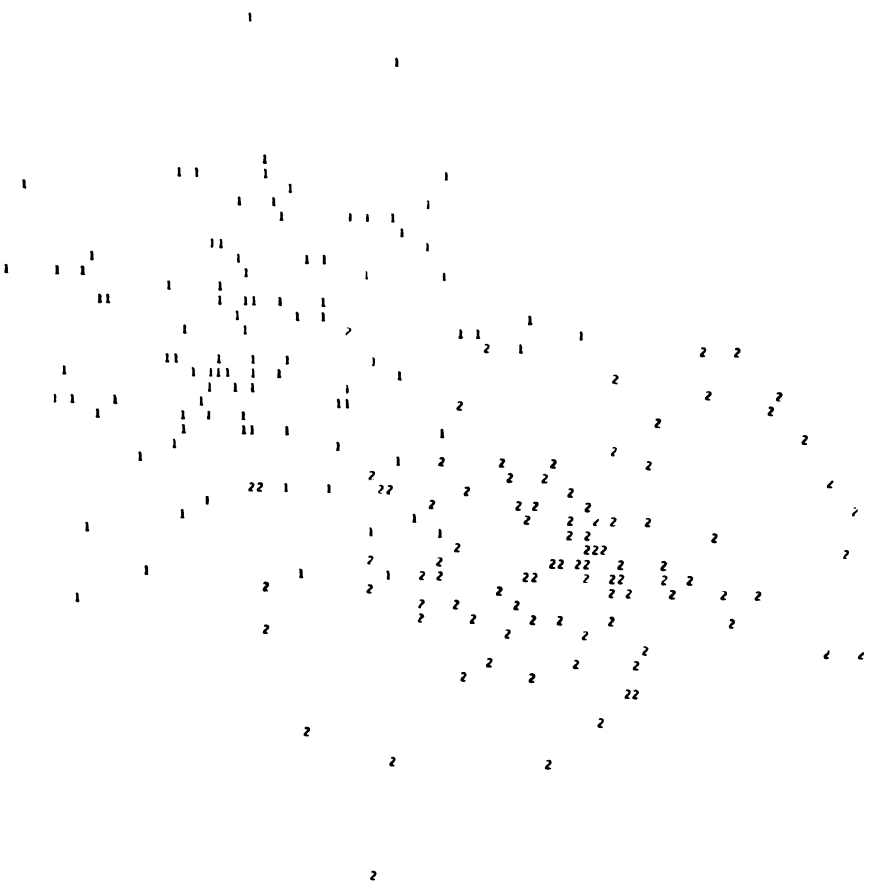


Figure 34E. Bivariate Gaussian Clusters.
Two clusters of D, as generated.

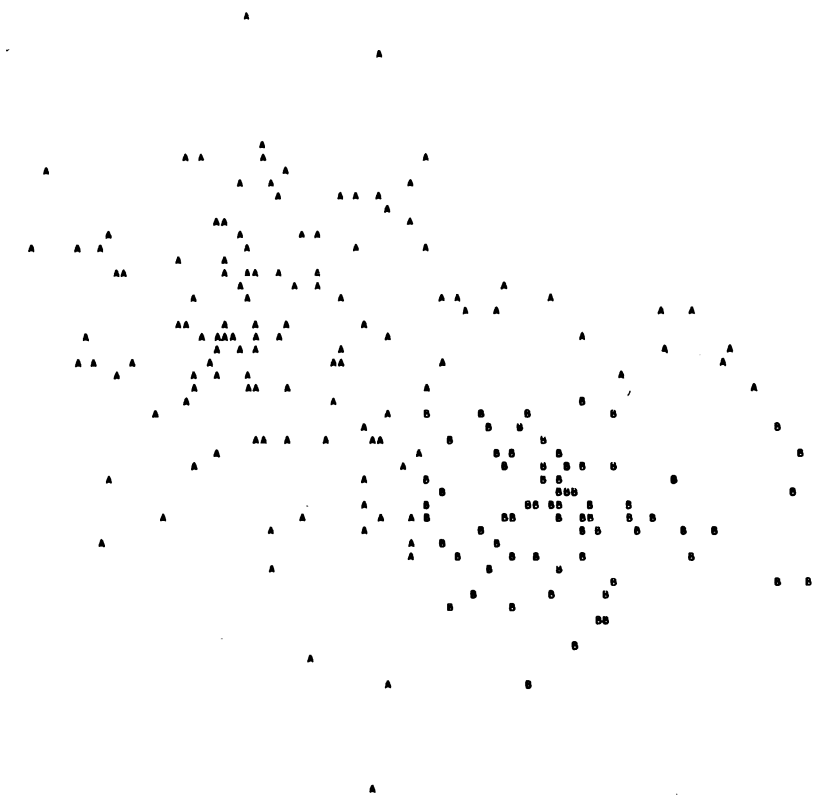


Figure 34F. Bivariate Gaussian Clusters.
Two clusters of D, as separated by the Unimodal Tree Algorithm.

Figure 34G. Bivariate Gaussian Clusters.
Two elliptical Gaussian clusters, unseparated.

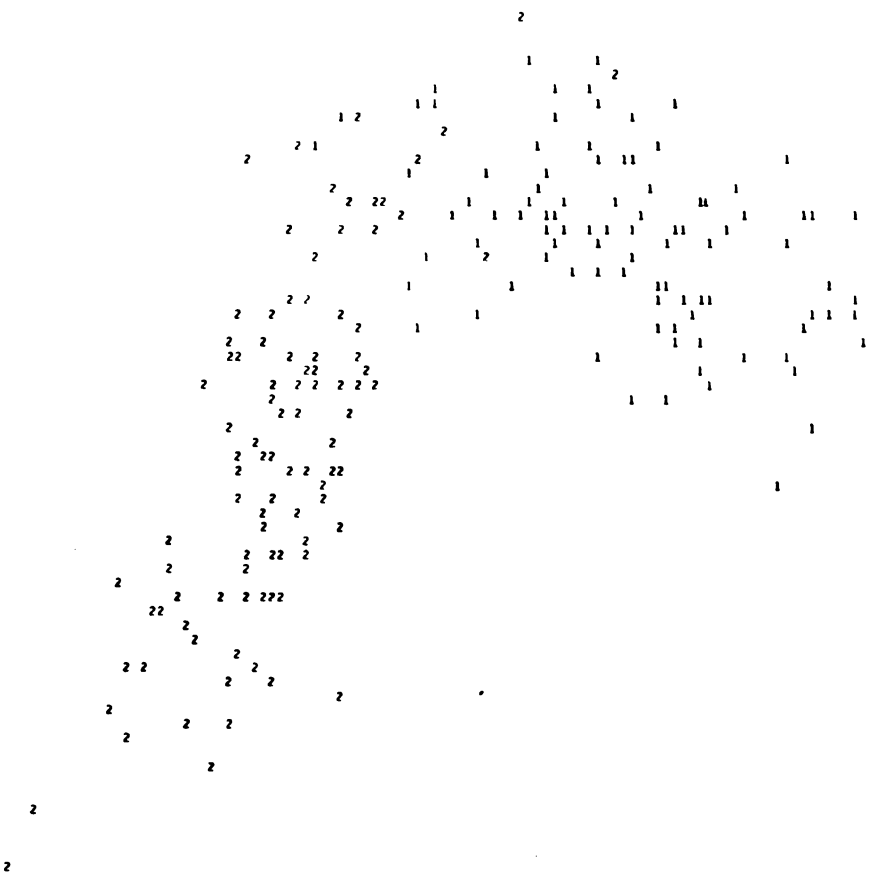


Figure 34H. Bivariate Gaussian Clusters.
Two elliptical Gaussian clusters, as generated.

Figure 34I. Two Elliptical Gaussian Clusters, as Separated by the Unimodal Tree Algorithm.

Figure 35A. Gaussian and Ring-Type Cluster, Unseparated.

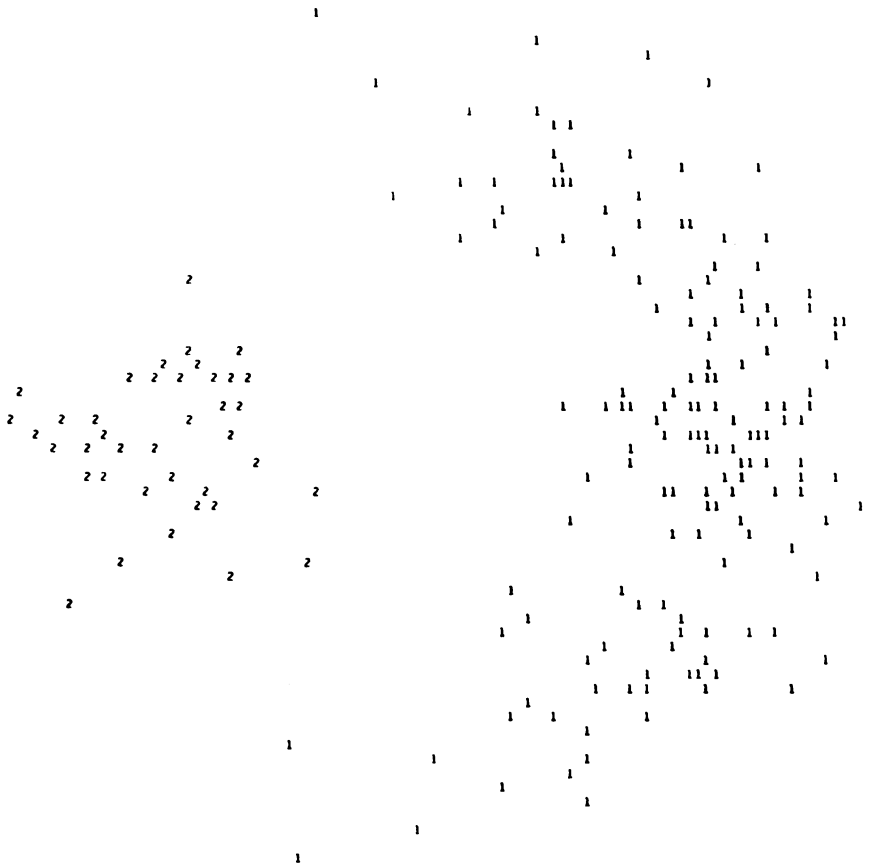


Figure 35B. Gaussian and Ring-Type Cluster, as Generated.

Figure 35C. Gaussian and Ring-Type Cluster, as Separated by the Unimodal Tree Algorithm.

demonstrates the independence of the clustering scheme to cluster shape.

5.5 Application to Optimization

Another possible application of the Unimodal Tree Algorithm is as an aid in finding the maxima and minima of a function of several variables in a region. If the smallest distance between two maxima which we desire to distinguish is known, a grid step size less than half of this distance may be used. The function is sampled at the points of this grid, and the resulting grid function is run through the Unimodal Tree Algorithm. A conventional optimization technique may begin at the highest point of each subset thus obtained. While the normal philosophy in procedures for finding a maximum of a function is to minimize the number of function evaluations [6], the procedure suggested here may be useful if it is suspected that more than one maximum might exist, and all the maxima are desired. The same procedure may be applied to the negative of a function to obtain its minima, or the algorithm may be appropriately modified.

CHAPTER VI

CONCLUSION

The major accomplishment of this thesis is the development of the Unimodal Tree Algorithm. This algorithm can be used in both a scheme for object isolation and a scheme for clustering of points. The algorithm requires approximately twice the amount of storage and computation when the number of grid points is doubled. In addition, the shapes of the objects and clusters which are generated may be quite general. The amount of computation and storage required is low enough that clustering and object isolation may be performed on a "mini"-computer such as the IBM 1800. The computer print outs shown in this thesis were generated using the IBM 1800 at Michigan State University.

Several attempts were made to develop an algorithm applicable to object isolation before the Unimodal Tree Algorithm was developed. These attempts each attacked certain deficiencies in their predecessors, but had deficiencies of their own. The Subset Uniting Procedure solved the problem of missing local maxima in part two of procedure F, but required an amount of computation roughly increasing with the cube of the number of subsets ($\sim\frac{n^2}{2}$ subset adjacency tests each requiring $\sim n$ tests of the ultrametric inequality) in the general case, or increasing roughly with the square of the number of subsets in the case of a grid

(the adjacency test simplifies to testing for two points, one in each subset, at distance one from each other). The Diamond-function Procedure reduced the storage requirements for part one of procedure F to a fixed amount, but inherited procedure F's difficulty with points of equal value. The Potential Local Maxima Procedure reduced computational requirements to being almost linear with the number of data points present, but generated spurious regions from "flat" areas of the input scene. The Unimodal Tree Algorithm has none of these deficiencies, as long as the input scene is preprocessed in a manner such that all the objects become unimodal regions.

As an object isolation procedure, the Unimodal Tree Algorithm has the disadvantage of requiring each object to form a unimodal subset. Small variations in the intensity of the input scene generate unimodal regions just as do large variations. Of course, these small variations can be "smoothed out" while the larger fluctuations are retained.

In a clustering scheme, the Unimodal Tree Algorithm has several disadvantages, the most important of which is the astronomically large number of grid points required for spaces of high dimensionality. Another disadvantage is the fact that uniform or nearly uniform clusters cause small random "hills" in the density function which are separated into clusters by the Unimodal Tree Algorithm. This disadvantage may be able to be overcome in some cases by the formation of a histogram of the density function values. Uniform clusters will generate "spikes" in this histogram. The uniform clusters may be made "flat" by setting

all the values in each spike to the mean of that spike. A further disadvantage of this scheme is that the parameters of grid size, w_1, w_2, and Nav must be selected before clustering is performed. It may be possible to select these parameters automatically based upon the data set.

Future work can be done in the application of the Unimodal Tree to object isolation, clustering, and optimization. Automatic selection of grid size and smoothing parameters in both object isolation and in clustering would eliminate the necessity of finding usable parameters by trial and error. In the clustering of regions having a uniform density, the histogram procedure suggested above may be tried. Touching uniform clusters might be separable if smoothing is applied after the histogram procedure. The narrow "neck" between two uniform clusters would tend to "erode" downward in value while the clusters of the uniform clusters would stay about constant in magnitude, thus generating one unimodal subset for each uniform cluster. It is hoped that the Unimodal Tree Algorithm will make object isolation and clustering involving general shapes practical for use on "mini"-computers as well as on large-scale machines.

REFERENCES

LIST OF REFERENCES

1. Mehdi Behzad and Gary Chartrand, Introduction to the Theory of Graphs, Allyn and Bacon, Boston. 1971.

2. A. T. Bertziss, Data Structure Theory and Practice, Academic Press, New York, pp. 190-191. 1971.

3. R. A. Fisher, "The Use of Multiple Measurements in Taxonomic Problems" Annals of Eugenics, Vol. 3, Part 2, pp. 179-188, 1936.

4. R. W. Floyd, "Algorithm 245: Treesort 3," Communications of the ACM, Vol. 7, No. 12, p. 701. December 1964.

5. Israel Gitman and Martin D. Levine, "An Algorithm for Detecting Unimodal Fuzzy Sets and Its Application as a Clustering Technique," IEEE Transactions on Computers, Vol. C-19, No. 7, pp. 583-593, July 1970.

6. Ronald L. Gue and Michael E. Thomas, Mathematical Methods in Operations Research, MacMillan. p. 104. 1970.

7. Steven C. Johnson, "Heirarchical Clustering Schemes," Psychometrika, Vol. 32, No. 3, 241-254, September, 1967.

8. G. Nagy, "State of the Art in Pattern Recognition," Proceedings of the IEEE, Vol. 56, No. 5, pp. 836-862, May, 1968.

9. L. A. Zadeh, "Fuzzy Sets," Information and Control, Vol. 8, No. 3, pp. 338-353, June 1965.

10. Charles T. Zahn, "Graph-Theoretical Methods for Detecting and Describing Gestalt Clusters," IEEE Transactions on Computers, Vol. C-20, No. 1, pp. 68-86, January, 1971.

APPENDIX

APPENDIX

FORTRAN LISTING OF THE UNIMODAL TREE ALGORITHM

```
      SUBROUTINE UNIMD(N,NDIM,IBNDS,CAND,NEXT)
C-----
C-----PROGRAMMED BY ROBERT WALTON, DECEMBER 1972.
C-----
C-----THIS SUBROUTINE SEPARATES A FUZZY SET WHOSE MEMBERSHIP VALUE IS IN ARRAY
C-----CAND IN A UNIFORM GRID INTO UNIMODAL FUZZY SETS.  THE NUMBER OF THE FUZZY
C-----SET TO WHICH EACH GRID POINT BELONGS IS RETURNED IN CAND
C-----ALL COMPUTATIONS ARE OF ORDER N
C-----THE MEMBERSHIP VALUE OF ALL POINTS MUST BE GREATER THAN ZERO
C-----THE CODING OF THE SUBROUTINE IS COMPLETELY MACHINE-INDEPENDENT
C-----THIS SUBROUTINE WILL RUN CORRECTLY USING ANY FORTRAN-II COMPILER
C-----
C-----SUBROUTINE UNIMD USES THREE OTHER SUBROUTINES FOR ADDRESSING AND INDEXING
C-----THE INPUT DATA POINTS IN THE SPACE DESIRED.
C-----SUBROUTINE COORD AND SUBROUTINE IADDR ACT AS MULTI-DIMENSIONAL
C-----SUBSCRIPTING SUBROUTINES.  COORD TAKES THE ADDRESS OF A DATA POINT IN
C-----THE LINEAR ARRAY CAND AND COMPUTES THE GRID COORDINATES IN AN NDIM-
C-----DIMENSIONAL GRID WITH IBNDS(J) GRID STEPS IN DIMENSION J.
C-----SUBROUTINE IADDR PERFORMS THE INVERSE OPERATION.  IADDR FORMS THE LINEAR
C-----ADDRESS OF THE POINT FROM ITS COORDINATES OR SUBSCRIPT NUMBERS.
C-----SUBROUTINE INCRM COMPUTES THE COORDINATES OF THE NEIHGBORS OF A POINT
C-----INPUT AS COORDINATE VALUES.  THE NUMBER OF NEIGHBORS IS DEPENDENT UPON
C-----THE DEFINITION OF THE GRAPH G.  THE NUMBER ALSO VARIES WITH THE NUMBER OF
C-----DIMENSIONS.  THE CURRENT VERSION IS WRITTEN FOR G HAVING EDGES BETWEEN
C-----PAIRS OF VERTICES HAVING EUCLIDEAN DISTANCE 1.
C-----THESE THREE SUBROUTINES ARE CURRENTLY WRITTEN FOR A SPACE OF ANY DIMENSION
C-----LESS THAN ELEVEN.  CONSIDERABLE SAVINGS IN COMPUTATION TIME MAY BE
C-----OBTAINED BY WRITING THESE SUBROUTINES SPECIFICALLY FOR THE NUMBER OF
C-----DIMENSIONS BEING USED, IF IT IS FIXED.  SAVINGS ARE ACCOMPLISHED BY
C-----ELIMINATING SUBSCRIPT OPERATIONS.  THE CALLS IN UNIMD MAY BE MODIFIED TO
C-----ACCOMPLISH THE INCREASE IN EFFICIENCY.
C-----THE PARAMETER SEQUENCES FOR THESE THREE SUBROUTINES AS THEY ARE WRITTEN IS
C-----    CALL COORD(CBT,IC,NDIM,IBNDS)
C-----    CALL INCRM(IX,IC,KK,NDIM,IBNDS,L)
C-----    CBT=IADDR(IX,NDIM,IBNDS)    ,
C-----WHERE CBT IS A LINEAR ADDRESS, IC AND IX ARE COORDINATE ARRAYS,
C-----NDIM IS THE NUMBER OF DIMENSIONS, IBNDS IS AN ARRAY HOLDING THE NUMBER OF
C-----GRID STEPS IN EACH OF THE NDIM DIMENSIONS, KK IS THE NUMBER OF THE
C-----NEIGHBOR POINT DESIRED, AND L IS ZERO IF THIS NEIGHBOR POINT IS IN BOUNDS,
C-----AND IS NON-ZERO IF THE NEIGHBOR IS OUT OF BOUNDS.
C-----
C-----CAND HOLDS THE INITIAL DATA, AND FINALLY THE GROUPS
C-----NEXT KEEPS POINTERS TO ALL POINTS WHICH HAVE HAD ASSIGNMENT WITH BRANCHING
C
      INTEGER CAND(1000),NEXT(1000)
C
C-----IC HOLDS THE COORDINATES OF THE CURRENT BRANCH POINT
C-----IX HOLDS THE COORDINATES OF EACH SUCCESSIVE PROSPECTIVE NEXT BRANCH POINT
C-----IBNDS HOLDS THE NUMBER OF GRID POINTS IN EACH DIMENSION
C
      INTEGER IC(10),IX(10),IBNDS(10)
C
C-----CBT HOLDS THE ADDRESS OF THE CURRENT BRANCH POINT, AND NBT THE ADDRESS OF
C-----THE NEXT BRANCH POINT
C
      INTEGER CBT,NBT
C-----
C-----COMPUTE THE NUMBER OF NEIGHBOR POSITIONS FROM EACH POINT IN THE GRID.
C-----THE VALUE OF NEIGH DEPENDS ON SUBROUTINE INCRM, AND MUST JIVE WITH THE
```

```
C-----NUMBER OF NEIGHBOR POINTS SUBROUTINE INCRM IS WRITTEN TO GENERATE.
      NEIGH=2*NDIM
C-----
C-----THE SIGN OF EACH DATA WORD IS USED AS AN ASSIGNMENT FLAG.  IF THE
C-----DATA WORD IS NEGATIVE, IT HAS NOT BEEN ASSIGNED YET.  IF IT IS
C-----POSITIVE, IT HAS BEEN ASSIGNED.  FOR THIS REASON, ALL THE DATA
C-----VALUES PRESENTED TO THIS PROGRAM MUST BE POSITIVE.
C-----
C-----SET FLAG FOR NO ASSIGNMENT YET
C-----TEST TO SEE IF ANY OF THE INPUT DATA IS NOT POSITIVE.
      DO 100 I=1,N
C-----GET THE INPUT FUNCTION VALUE FOR POINT I
      MM=CAND(I)
C-----TEST TO BE SURE IT IS POSITIVE
      IF(MM)540,540,100
C-----IF A MEMBERSHIP VALUE WAS OUT OF RANGE, STOP 1 IS EXECUTED
540   STOP 1
C-----OTHERWISE, THE SIGN IS CHANGED.
100   CAND(I)=-MM
C-----
C-----IG WILL BE THE NUMBER OF GROUPS DEFINED
      IG=0
C-----THE NEXT SECTION FINDS AN ARBITRARY POINT TO BEGIN GROUP IG WITH.
C-----THE POINT SELECTED WILL BE THE HIGHEST UNASSIGNED POINT (OR ONE OF THEM,
C-----IN CASE OF TIES)
C-----NOTE THAT ANY UNASSIGNED POINT COULD BE SELECTED HERE.  CHOOSING THE
C-----HIGHEST UNASSIGNED POINT WAS AN ARBITRARY DECISION ON MY PART.
C-----THE ADVANTAGE TO CHOOSING THE HIGHEST POINT TO START EACH GROUP IS THAT
C-----THE HIGHEST GROUPS WILL HAVE THE MAXIMUM NUMBER OF MEMBERS.
C-----INITIALLY, SET THE MAXIMUM VALUE YET FOUND TO ZERO
542   MAX=0
C-----SET THE CURRENT BRANCH POINT TO ZERO AS A FLAG, IN CASE THERE ARE NO
C-----UNASSIGNED POINTS LEFT
      CBT=0
C-----SEARCH ALL POINTS FOR THE HIGHEST UNASSIGNED POINT
      DO 101 II=1,N
C-----GET THE VALUE OF POINT II
      MM=CAND(II)
C-----TEST FOR NOT ASSIGNED YET (SIGN BIT ON)
      IF(MM)500,101,101
C-----IF UNASSIGNED, GET THE MAGNITUDE OF THE POINT
500   MM=-MM
C-----TEST TO SEE IF HIGHER THAN ANY POINT YET ENCOUNTERED
      IF(MM-MAX)101,101,501
C-----IF SO, RESET THE HIGHEST MAGNITUDE YET FOUND AND ASSIGN CBT TO THIS POINT
501   MAX=MM
      CBT=II
101   CONTINUE
C-----TEST IF A CBT HAS BEEN FOUND.  IF NOT, ALL POINTS HAVE BEEN ASSIGNED
      IF(CBT)543,543,544
C-----
C-----IN THIS SECTION, A UNIMODAL TREE IS FOUND USING THE STARTING POINT FOUND
C-----IN THE PREVIOUS SECTION
C-----MODE = VALUE OF HIGHEST POINT YET FOUND IN THIS UNIMODAL TREE.  START MODE
C-----AT THE VALUE OF THE FIRST POINT OF THE TREE
544   MODE=-CAND(CBT)
C-----SET THE ASSIGNMENT FLAG FOR THIS 1ST POINT OF THIS NEW GROUP
      CAND(CBT)=MODE
C-----NEXTC COUNTS THE NUMBER OF BRANCHING POINTS WHICH HAVE BEEN ENCOUNTERED
```

```
      NEXTC=0
C-----NEXTO COUNTS THE NUMBER OF POINTS IN THE NEXT TABLE WHICH HAVE BEEN
C-----RE-CHECKED FOR ADDITIONAL BRANCHES.  NEXTO POINTS TO THE LAST ENTRY
C-----WHICH HAS BEEN CHECKED IN THE NEXT TABLE.
      NEXTO=0
C-----UP THE NUMBER OF GROUPS BY ONE
      IG=IG+1
C-----HERE DETERMINE THE INITIAL MAGNITUDE OF THIS LINK
502   MAG=CAND(CBT)
C-----COMPUTE COORDINATES OF CURRENT BRANCH POINTS IN ARRAY IC
      CALL COORD(CBT,IC,NDIM,IBNDS)
C-----GO THROUGH THE POTENTIAL NEIGHBOR POSITIONS
C-----THERE ARE NEIGH OF THESE NEIGHBOR POSITIONS
      DO 104 KK=1,NEIGH
C-----PUT THE COORDINATES OF THE KK-TH NEIGHBOR POINT IN ARRAY IX
C-----L WILL BE RETURNED NON-ZERO IF NEIGHBOR POINT KK IS OUT-OF-BOUNDS
      CALL INCRM(IX,IC,KK,NDIM,IBNDS,L)
C-----CHECK TO SEE IF IN BOUNDS
      IF(L)104,511,104
C-----IF SO, GET ADDRESS OF NEIGHBOR POINT
511   NBT=IADDR(IX,NDIM,IBNDS)
C-----GET VALUE OF NEIGHBOR
      MM=CAND(NBT)
C-----SEE IF THIS NEIGHBOR CAN BE ASSIGNED
C-----TEST FOR ALREADY ASSIGNED
      IF(MM)518,104,104
C-----TEST FOR RISE, FALL, OR EQUALITY
518   IF(MAG+MM)520,513,513
C-----IF RISE, BE SURE MODE OF GROUP HAS SAME MAGNITUDE AS THE CURRENT
C-----BRANCH POINT
520   IF(MODE-MAG)530,515,104
C-----THERE HAS BEEN AN ERROR IF THE CURRENT BRANCH POINT IS HIGHER THAN THE
C-----MODE OF THE GROUP.  IF THIS IS THE CASE, EXECUTE STOP 2.
530   STOP 2
104   CONTINUE
C-----THE DO-LOOP FALLS THROUGH ONLY IF NONE OF THE NEIGHBOR POINTS WERE
C-----ABLE TO BE ASSIGNED
C-----IN THIS CASE, GO PICK UP AN OLD BRANCH POINT FROM LIST NEXT AND
C-----RE-CHECK IT FOR ADDITIONAL BRANCHES
      GO TO 507
C-----IF AN ASSIGNABLE NEXT BRANCH POINT WAS FOUND AND IT WAS HIGHER THAN THE
C-----CURRENT MODE OF THE GROUP, RESET THE MODE OF THE GROUP TO THIS POINT
515   MODE=-MM
C-----IF AN ASSIGNABLE NEXT BRANCH POINT WAS FOUND, SET ITS ASSIGNMENT FLAG
513   CAND(NBT)=-MM
C-----ENTER THE CURRENT BRANCH POINT INTO TABLE NEXT AND CONTINUE THIS BRANCH
C-----OF THE UNIMODAL TREE WITH THE NEXT BRANCH POINT (NBT)
C-----INCREMENT NEXTC
      NEXTC=NEXTC+1
C-----SET NEXT(NEXTC) TO THE CURRENT BRANCH POINT
      NEXT(NEXTC)=CBT
C-----SET THE CURRENT BRANCH POINT TO THE NEXT BRANCH POINT
      CBT=NBT
C-----GO BACK AND PERFORM ANOTHER LINKING TO THE NEW CBT, IF POSSIBLE
      GO TO 502
C-----IF IT WAS NOT POSSIBLE TO ASSIGN ANY NEIGHBOR POINTS, ASSIGN THE CURRENT
C-----BRANCH POINT TO THE GROUP CURRENTLY BEING FORMED.
507   CAND(CBT)=IG
C-----HERE CHECK IF THERE ARE ANY UNPROCESSED OLD BRANCH POINTS LEFT
```

```
C-----INCREMENT THE NEXT-OUT POINTER.
      NEXTO=NEXTO+1
      IF(NEXTC-NEXTO)542,509,509
C-----IF THERE ARE, PICK UP THE NEXT ONE AS THE CURRENT BRANCH POINT AND CHECK
C-----ITS NEIGHBORS
509   CBT=NEXT(NEXTO)
C-----GO ATTEMPT TO EXTEND THIS BRANCH
      GO TO 502
C-----WHEN THERE ARE NO POINTS LEFT, RETURN.  A MAXIMAL UNIMODAL PARTITIONING
C-----OF THE INPUT DATA IS NOW IN ARRAY CAND.
543   RETURN
      END
      SUBROUTINE INCRM(IX,IC,KK,NDIM,IBNDS,L)
      INTEGER IX(10),IC(10),IBNDS(10)
C-----THIS SUBROUTINE INCREMENTS THE ADDRESS GIVEN TO IT IN IC TO ARRAY IX.
C-----THE KK-TH NEIGHBOR POINT IS ENTERED INTO IX.
C-----IF POINT IX IS IN BOUNDS, L=0.  IF IX IS OUT OF BOUNDS, L = 1
C-----THIS PARTICULAR VERSION OF INCRM IS WRITTEN FOR AN NDIM-DIMENSIONAL GRID
C-----WITH GRAPH G HAVING EDGES BETWEEN POINTS OF DISTANCE 1.
      L=0
C-----OBTAIN DIMENSION TO BE INCREMENTED
      J=(KK+1)/2
C-----OBTAIN NUMBER (+ OR - 1 ) TO INCREMENT DIMENSION BY
      K=(KK-J*2)*2+1
C-----COPY THE CENTER POINT'S COORDINATES
      DO 100 I=1,NDIM
100   IX(I)=IC(I)
C-----INCREMENT THE PROPER DIMENSION BY THE PROPER AMOUNT
      IX(J)=IX(J)+K
C-----TEST TO SEE IF OUT OF BOUNDS
      IF(IX(J))500,500,501
501   IF(IX(J)-IBNDS(J))502,502,500
500   L=1
502   RETURN
      END
      FUNCTION IADDR(IX,NDIM,IBNDS)
      INTEGER IX(10),IBNDS(10)
C-----IADDR COMPUTES THE ADDRESS OF A POINT GIVEN ITS VECTOR OF LOCATION
      IADDR=1
      K=1
C-----UN-SUBSCRIPT THE COORDINATES INTO THE ADDRESS THEY INDICATE
C-----THIS MAKES CAND LIKE AN NDIM-DIMENSIONAL ARRAY
      DO 100 I=1,NDIM
      IADDR=IADDR+(IX(I)-1)*K
      K=K*IBNDS(I)
100   CONTINUE
      RETURN
      END
      SUBROUTINE COORD(L,IC,NDIM,IBNDS)
      INTEGER IC(10),IBNDS(10)
C-----SUBROUTINE COORD COMPUTES THE COORDINATES OR LOCATION OF A POINT GIVEN ITS
C-----ADDRESS
      M=L
      J=NDIM-1
      K=1
C-----COMPUTE LARGEST BLOCK SIZE
      DO 100 I=1,J
100   K=K*IBNDS(I)
C-----GO THROUGH AND COMPUTE THE COORDINATES OR SUBSCRIPTS OF THE INPUT
```

```
C-----POINT.   THIS ROUTINE IS THE INVERSE OF IADDR
      DO 101 I=1,NDIM
      J=NDIM -I+1
      IC(J)=(M-1)/K+1
      M=M-(IC(J)-1)*K
C-----WATCH OUT WE DON'T DIVIDE BY IBNDS(0)
      IF(J-1)101,101,500
500   K=K/IBNDS(J-1)
101   CONTINUE
      RETURN
      END
```