A Hybrid Methodology for Data Clustering

Eric William Tyree

Submitted to the Department of Computer Science
City University
for the degree of
Doctor of Philosophy
October, 1998

Department of Computer Science
City University
Northampton Sq.
London, EC1V 0HB

SYNOPSIS

This thesis introduces and evaluates a new hybrid method for the searching for groups in data - a process referred to as cluster analysis. The Agglomerative - Partitional Clustering methodology (APC) proposed in this work is a novel solution to the clustering problem intended for use with large, noisy data sets and capable of recovering clusters of arbitrary shape.

Large sample size, noise and nonhyperellipsoidal cluster shapes can create difficulties for many clustering algorithms. Many commonly used clustering techniques are too inefficient to handle large data sets found in many data analysis problems or are limited by the fact that they implicitly or explicitly define clusters as being hyperellipsoidal (*i.e.* "globular" in shape) and can therefore fail to recover other types of cluster structure. Moreover, the presence of noise can also make detection of cluster structures problematic, particularly for clustering techniques that are explicitly designed to handle nonhyperellipsoidal cluster structures.

APC is able to circumvent these difficulties by hybridising a number of diverse approaches to clustering. Large data sets are dealt with by hybridising fast pattern partitioning techniques with hierarchical and density search methods. Arbitrary cluster shapes are handled by a unique linked line segment representation of cluster shape. In short, rather than representing clusters with their centroids, the clusters are represented via a piecewise linear approximation of the cluster structure. This enables APC to represent any cluster shape that is piecewise linearly approximatable.

The purpose of this thesis, therefore, is to introduce APC and to evaluate the ability of APC to recover cluster structure under the conditions described above. First, it is

argued that there is a dearth of clustering techniques that can process large, noisy data sets where there exists arbitrarily shaped clusters. Next, the APC approach to clustering is described in detail. Here it is discussed how APC is able to handle voluminous and noisy data without being constrained to any particular cluster shapes. Moreover, as APC represents a hybridisation of clustering strategies and techniques, different ways of implementing APC are also evaluated.

The remainder of this thesis is concerned with the evaluation of APC. First, APC is empirically compared to other clustering methods via Monte Carlo simulation on a number of complex data sets. A wide variety of experimental conditions examining cluster shape, dispersion, noise and dimensionality are covered. The use of APC as a data reduction method is also examined. This final experiment also highlights the utility of the linked line segment representation of cluster shape proposed in this thesis.

Finally, the concluding chapter summarises the results and limitations of this thesis and discusses some future directions this research could take.

List of Figures

List of Tables

Acknowledgements

I would like to thank all who have helped, encouraged and supported me throughout the development of this thesis. Particularly Dr. J. A. Long for his supervision, friendship and for giving me the freedom to explore my research interests without undue restraint. I would also like to extend my gratitude to my colleagues at Searchspace and the Department of Computer Science for their support and encouragement.

Most of all I would like to thank Veronica for her love, support and patient understanding without which none of this would have been possible.

Declaration

Table of Abbreviations

Term	**Definition**
ADD	Absolute Density Distance
ALINK	Average Linkage
APC	Agglomerative-Partitional Clustering
ARI	Adjusted Rand Index
CLINK	Centroid Linkage
KSONN	Kohonen Self-Organising Neural Network
PCA	Principal Component Analysis
SLINKS	Single Linkage
SOM	Self-Organizing Map
UDD	Uniform Density Distance
WARD	Ward's Method
WHM	Wong's Hybrid Method
WTANN	Winner Take All Neural Network

1. Introduction

The first stage in analysing complex data sets is often the application of any number multivariate statistical techniques in order to gain an understanding of how the data is structured. This information can in turn be used for a variety of tasks such as data reduction, feature extraction and exploratory data analysis. A wide range of techniques exist (see Krzanowski (1993) for review) that can be utilised for accomplishing these tasks such as those that look for principal axes in the data (*e.g.* linear and non-linear principal components analysis) and data visualisation methods such as Andrew plots, glyphs and Chernoff plots. Other techniques concentrate on extracting homogenous groupings or clusterings within the data, an approach to data analysis referred to as cluster analysis. The purpose of this thesis is to introduce and evaluate a new method for performing cluster analysis on numerical data sets.

Although many of the clustering techniques in common use today were initially developed in the 1960's and 1970's there are still some aspects of the clustering problem worthy of continued research. For example, given the continuing development of cost effective methods of data storage, there is a need for clustering techniques that can analyse large quantities of data quickly and efficiently. Many of the clustering techniques developed in the last few decades are too inefficient to handle large data sets found in many data analysis problems.

Another difficult problem in cluster analysis is the detection and recovery of nonhyperellipsoidal cluster shapes. If arbitrarily shaped structures exist in a given data set, it is imperative that the clustering techniques being utilised to analyse the data are capable of extracting these structures or important information contained the data will be lost. Many commonly used clustering techniques are limited by the fact that they implicitly or explicitly define clusters as being hyperellipsoidal (*i.e.* "globular" in shape) and therefore can fail to recover other types of cluster structure.

The presence of noise can also make detection of cluster structures problematic, particularly for clustering techniques that are explicitly designed to handle nonhyperellipsoidal cluster structures. Noise can arise from a variety of sources such as measurement error, data corruption, missing attributes or poor cluster separation. Many techniques such as single linkage (McQuitty, 1957; Sneath, 1957) and Wong's hybrid method (Wong, 1982) that are capable of extracting arbitrarily shaped clusters do not perform well under noisy conditions[1].

The Agglomerative - Partitional Clustering methodology (APC) proposed in this work is a novel solution to the clustering problem intended for use with large, noisy data sets and capable of detecting clusters of arbitrary shape. The primary purpose of this thesis, therefore, is to introduce APC and evaluate its ability to cluster data under the above conditions.

The remainder of this chapter is organised as follows: sections 1.1 and 1.2 give a brief overview of cluster analysis and discuss the motivation of this thesis - the problem of applying clustering methods to large, noisy data sets with arbitrarily shaped clusters. Section 1.3 provides a general outline of APC. Sections 1.4 - 1.6

[1]See Arabie and Hubert (1996) for other unresolved problems in cluster analysis.

discuss the objectives, original contributions and scope of this thesis followed by section 1.7 which provides an overview of the thesis organisation.

1.1 Background: Cluster Analysis

Cluster analysis refers to the placing of observations or patterns into groups or classes such that members within one class are more similar to one another in some way than they are to members of other classes (Everitt, 1981)[2]. Simply stated, cluster analysis is the grouping or classification of previously unclassified data. More formally, let $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots , \mathbf{x}_N\}$ be a set of d-dimensional data vectors and $\boldsymbol{C} = \{C_1, C_2, \dots , C_k\}$ be a set of groups or classifications. Clustering techniques attempt to classify each of the N data vectors into at least one of the k groups. The classification of previously unclassified data is also sometimes referred to as the clustering problem.

Cluster analysis is also often referred to as *unsupervised learning* or *unsupervised classification* and should be differentiated from *supervised* learning or classification techniques. In cluster analysis the groupings within the data are unknown *a priori*. In other words, cluster analysis techniques are for the grouping or classification of previously unclassified data where there is no dependent variable to guide the construction of the classification model. In this sense cluster analysis can be viewed as a class of techniques for exploratory data analysis although it does have important uses in data reduction and feature extraction (see Hand, 1981).

Most clustering techniques, referred to as *hard* clustering, only allow mutually exclusive groupings where each pattern belongs to one and only one group or

[2]It should be made clear at this point that the terms *observations* and *patterns* are used synonymously throughout this thesis. An observation or pattern refers to an individual element or subject of a data set. For example, if the data consists of the height and weight of ten people, an observation or pattern is one subject's height and weight.

classification. *Fuzzy* clustering (*e.g.* Bezdeck, 1981) on the other hand allows individual patterns to have multiple group memberships. In this case individual observations have a degree of membership in each group rather than an absolute membership in one group as in hard clustering. As this thesis is primarily concerned with a new approach to the hard clustering of patterns, fuzzy clustering will not be treated any further.

Generally speaking, clustering techniques can be categorised as being hierarchical or pattern partitioning (Backer, 1995; Gordon, 1981; Everitt, 1981)[3]. Hierarchical clustering methods themselves can be further subdivided into agglomerative and divisive techniques. Agglomerative hierarchical techniques begin by treating each observation as an individual cluster (see Everitt, (1993) for review). Each cluster (which can be an individual observation) is then agglomerated with its closest neighbouring cluster. The measure of "closeness" or distance between clusters is what often differentiates the various hierarchical clustering algorithms from each other (Lance and Williams, 1967a). Regardless of how intercluster distance is measured, the agglomeration process is repeated until the entire data set is grouped as a single cluster. The result is a complete hierarchical classification of the entire data set. Divisive hierarchical techniques (*e.g.* Wang *et al,* 1996; Lambert and Williams, 1966; MacNaughton-Smith *et al,* 1964; Fielding, 1977) initially treat the entire data set as a single cluster which is then recursively divided up into smaller clusters until each pattern is treated as a single cluster in its own right.

Pattern partitioning techniques attempt to subdivide the entire data set into a fixed number of clusters and are characterised by the lack of hierarchical relationships

[3]The above taxonomic scheme is used primarily for explanatory convenience. Other taxonomies of clustering algorithms have been proposed. For example, some treat hierarchical, pattern partitioning, graph theoretical techniques, density searching methods and clumping techniques that allow overlapping clusters as all being distinct types of approaches to clustering. Moreover, these various methods are not necessarily mutually exclusive. For example, density seeking algorithms can be hierarchical as is the case with Wong's hybrid method (Wong, 1982) or partitional as in the case of moment preserving (Liu and Tsai, 1989). See Backer (1995), Gordon (1986), Everitt (1980), Blashfield (1976) and Lorr (1983) for other taxonomies.

between clusters being assumed or imposed. The partitioning of patterns into clusters is often accomplished via the optimisation of an objective function. A common procedure is to move patterns from one cluster to another in order find a *k* cluster partitioning that optimises the objective function. A wide variety of function optimisation techniques such as steepest descent, simulated annealing and evolutionary approaches can be applied to pattern partitioning (*e.g.* Srikanth *et al,* 1995; Qiu *et al,* 1994; Babu and Murty, 1994, 1993; Zhang and Boyle, 1991; Selim and Alsultan, 1991; Forgy, 1965).

1.2 Motivation

Although there exists a wide range of clustering techniques, many methods only work well under certain conditions or are limited in some other respect. Sample size, cluster shape and level of noise present in the data can all have profound effects on the performance of many clustering methods (*e.g.* Tyree and Long, 1998, 1997, 1996; Mangiameli *et al,* 1996; Balakrishnan *et al,* 1994; Cowgill, 1993; Milligan, 1980, 1981a; Bayne *et al*, 1980).

Large sample sizes can discourage the use of many clustering methods, particularly those that require the calculation of proximity matrices consisting of all interobservation distances. Given the decreasing costs of data storage there is a need for clustering techniques that can analyse large quantities of data quickly and efficiently. Large data sets can be easily dealt with by k-means (*e.g.* Forgy, 1965), moving methods (*e.g.* Ismail and Kamel, 1989) and Wong's Hybrid Method (Wong, 1982) all of which are reasonably fast clustering algorithms with low memory requirements. Many hierarchical clustering algorithms, however, require the calculation of a $N(N - 1)/2$ element distance matrix which for large data sets is not practical. In addition, the necessity of calculating the distance between all observations and clusters can carry an unacceptably high computational cost in large data sets[4]. Even more recent implementations requiring *$O(N^2)$* CPU time and *$O(N)$* storage space or *$O(N)$* CPU time with *$O(N^2)$* storage (see Murtagh, (1992, 1983) for reviews) can still be impractical for larger data sets.

Many clustering techniques also have difficulty dealing with nonhyperellipsoidal clusters. If other cluster structures exist in a given data set, it is imperative that the clustering techniques being utilised can model such relationships or important information contained the data will be lost. Arbitrarily shaped clusters can be modelled by single linkage, minimum spanning tree based methods (*e.g.* Zahn, 1971) and Wong's Hybrid Method (WHM). However, many other commonly used clustering techniques are unable to adequately model arbitrary cluster shapes. For example, k-means, moving methods and other squared error optimisation techniques have a tendency to find hyperellipsoidal clusters (Cormack, 1971). Average linkage and most standard hierarchical methods (other than single linkage) have also been shown to have difficulties with chaining clusters (*e.g.* Tyree and Long, 1997, 1996; Lorr, 1983)

The presence of noise can also make detection of such structures problematic particularly for clustering techniques that can handle chaining structures (Backer, 1995; Milligan and Cooper, 1987; Milligan, 1981a, 1980)[5]. Noise can arise from a variety of sources such as measurement error, data corruption, missing attributes and uncorrelated structure in the data arising from unattributable sources. Techniques such as single linkage, WHM and minimum spanning tree based methods (MST) that can handle arbitrary cluster shapes do not perform well under noisy conditions. This is because the "chaining" tendencies that allow them to model arbitrarily shaped

[4]The initial distance matrix between all patterns requires $N(N - 1)/2$ distance calculations. A 5000 pattern data set would therefore require (5000 * 4999)/2 = 12,497,500 distance calculations.

[5]*Chaining* clusters are defined in chapter 2.

clusters can also cause these methods to falsely agglomerate distinct clusters under noisy conditions or when clusters are closely proximate.

Problems can become particularly acute when all three conditions: large sample size, nonhyperellipsoidal cluster shapes and high levels of noise are present in a given data set. As just mentioned, single linkage, MST and WHM which can model clusters of most any shape, perform poorly under noisy conditions or when clusters are poorly separated. Even worse, single linkage and most other standard hierarchical techniques are not applicable to large data sets. Although k-means is probably the fastest of all clustering techniques and has often found to be one of the best performing techniques in comparison studies (Balakrishnan *et al*, 1996, 1994; Milligan and Cooper, 1987; Milligan, 1981a; Bayne *et al*, 1980) it tends to find clusters of hyperellipsoidal shape and therefore may not adequately reflect the true structure of data if there are structures that are significantly nonhyperellipsoidal. What is needed are clustering methods that are reasonably fast computationally, can model clusters of arbitrary shape and are robust under noisy conditions.

The relevance of this research to modern data analysis can be illustrated through problems encountered in geodemographics and the banking industry. Openshaw (1995) gives a good discussion of the application of cluster analysis to UK census data. Openshaw outlines some technical problems particular to census data that can effect the performance of clustering techniques. Some of these problems include nonnormality, nonlinear relationships, variable size and homogeneity within geographical entities and high noise levels due to both measurement error and deliberate introduction by census takers to protect confidentiality. Bearing this in mind, Openshaw recommends that clustering techniques applied to census data should posses the following properties:

- Ability to allow the data to "speak for it self" and not impose any particular cluster structures (shapes) on the data.
- Possess robust performance under noisy conditions.
- Have sufficient flexibility to uncover whatever structure actually exists in the data.
- Some innate capability to discover the appropriate number of clusters in the data.

To this one could also add the ability to efficiently process large quantities of data. For example, a 60 economic variable data set covering Scotland at the post code level translates into almost 100 megabytes of data. In cases such as this, computational efficiency is obviously an important factor.

Another example of the need for fast and robust clustering techniques can be found in retail banking. A large national bank may use data sets on the order of 10 million records for customer profiling purposes (Adriaans and Zantinge, 1996). Given that banking habits many not necessarily be modelled as simple, relatively noise free linear relationships, there is clearly a need for faster and more sophisticated tools for dealing with this type of information.

1.3 The APC Methodology

APC offers a simultaneous solution to dealing with large, noisy data sets with cluster structures of arbitrary shape. It is reasonably fast computationally, thus allowing it to handle large data sets, can both detect and represent clusters of arbitrary shape and is fairly robust under noisy conditions. To accomplish this, APC combines together a number of clustering strategies hybridising hierarchical, pattern partitioning and density seeking clustering techniques.

APC enables the modelling of clusters of arbitrary shape in the pattern space by generating a connected graph of the centroids of clusters found in an initial partitioning of the data set. The analyst is given a hierarchical clustering of the initial clusters where the distance between clusters in the hierarchy is based strictly on the relative density of the regions that separate the initial cluster centroids. The newly agglomerated clusters are then modelled as line segments within the *d*-dimensional pattern space rather than as a list of centroids or as a new centroid corresponding to the agglomerated cluster. It will be argued that this representation allows more accurate measurement of the cluster membership individual members of nonhyperellipsoidal clusters have than centroids alone.

APC can be summarised as a four step process. First, an initial cluster analysis is run on the data using a pattern partitioning technique such as k-means, moving methods or unsupervised neural networks to partition the data into *k* subsets (clusters). The purpose of this stage is to identify regions of high density in the pattern space as in Wong (1982). The second step is to calculate an estimate of the density of the regions between the centroids of the initial clusters to be used as a measure of intercluster distance. Unlike standard hierarchical methods that often incorporate correlation or Minkowsky based metrics[6] in measuring intercluster distance, APC measures intercluster distance as being proportional to the density of patterns between clusters. The basic idea is that if two clusters found in an initial partitioning of the data coexist within a contiguous region of the pattern space that is of relatively high density, the two clusters should be agglomerated.

[6]Minkowsky metrics are defined as :

$$d^{\lambda}(\mathbf{x}_i,\mathbf{x}_j)=\left[\sum_{p=1}^{d} w_p\left|\mathbf{x}_{ip}-\mathbf{x}_{jp}\right|^{\lambda}\right]^{1/\lambda}$$

($\lambda > 0$). Two common examples of Minkowsky metrics are Euclidean and City Block (absolute value) which correspond to λ = 2 and λ = 1 respectively.

Once all of the intercluster distances have been calculated, the third step involves using the resulting $k(k - 1)/2$ element distance matrix to generate a connected graph of the initial clusters where the edge weights of the graph correspond to the distance between clusters. This graph can then be converted into a hierarchical tree (dendrogram) where the base consists of each of the initial clusters found in the initial pattern partitioning stage and the top of a single cluster grouping the entire data set. However, because APC limits agglomerations between initial clusters found in the pattern partitioning to neighbouring clusters, the reduction of the entire data set to a single agglomerated cluster cannot be guaranteed (see Chapter 4). In any case, by using a connected graph approach to cluster agglomeration, arbitrary cluster shapes can be modelled in a piecewise linear manner.

Finally, once a suitable cut-off point has been determined in the dendrogram, the agglomerated clusters consisting of the initial pattern partitioning stage clusters are represented by line segments joining the initial clusters together at the centroids. The membership of individual patterns to the cluster they belong to can then be measured as the distance between the pattern and the nearest line segment in the cluster belonging to the pattern. The APC methodology is outlined graphically in Figure 1.1.

The purpose of the linked line segment representation of cluster structure is to give a more faithful indication of the distance of individual patterns from a cluster. During subsequent analysis the agglomerated clusters are *not* represented with a new mean or centroid vector but with the linked line segments which can take into account irregular shapes. It will be argued that the use of centroids as representations of clusters that are highly nonglobular in shape leads to a loss of information regarding measurements of the cluster membership of individual patterns or observations. This information loss can have detrimental effects on subsequent analysis of individual observations - particularly when cluster analysis is used as a method of feature extraction or data reduction.

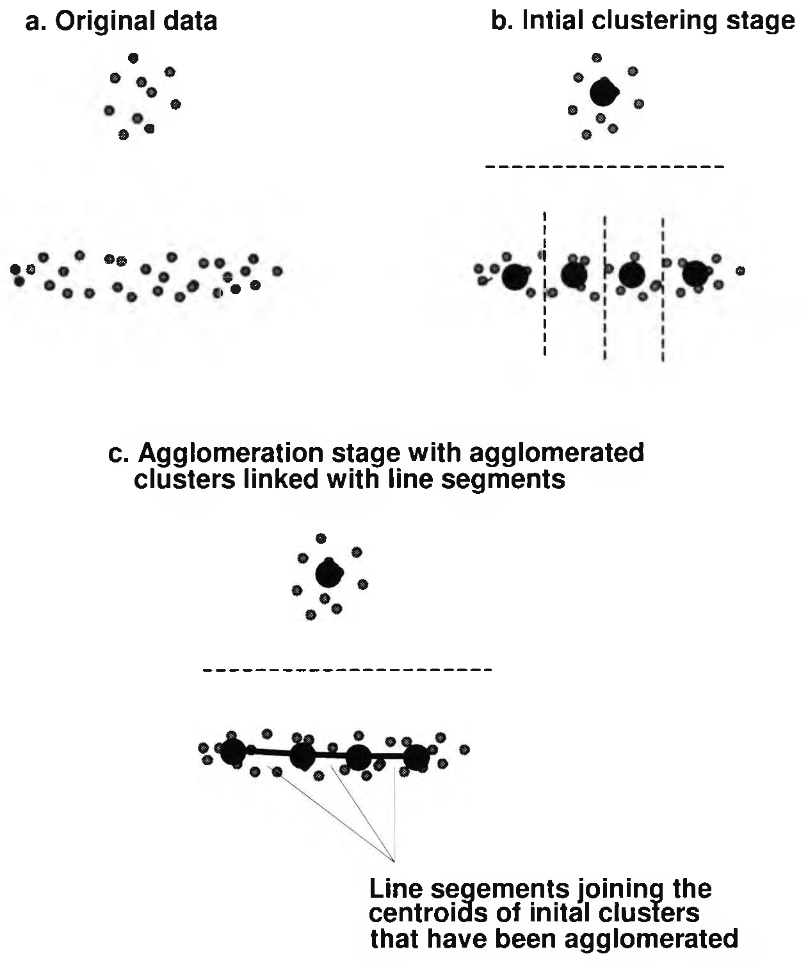

a. Original data
b. Intial clustering stage
c. Agglomeration stage with agglomerated clusters linked with line segments
Line segements joining the centroids of inital clusters that have been agglomerated

Figure 1.1: APC begins with an initial clustering of the data (a) using any standard pattern partitioning algorithm (b). The centroids of the resulting clusters (separated by dashed lines) are then presented to the agglomeration stage. The agglomeration stage (c) seeks to agglomerate initial clusters that are found in contiguous regions of high density as in the case of the elongated cluster at the bottom. Once agglomerated, the structure of the agglomerated clusters is represented by joining together with line segments the centroids of the initial clusters that are agglomerated into the new cluster.

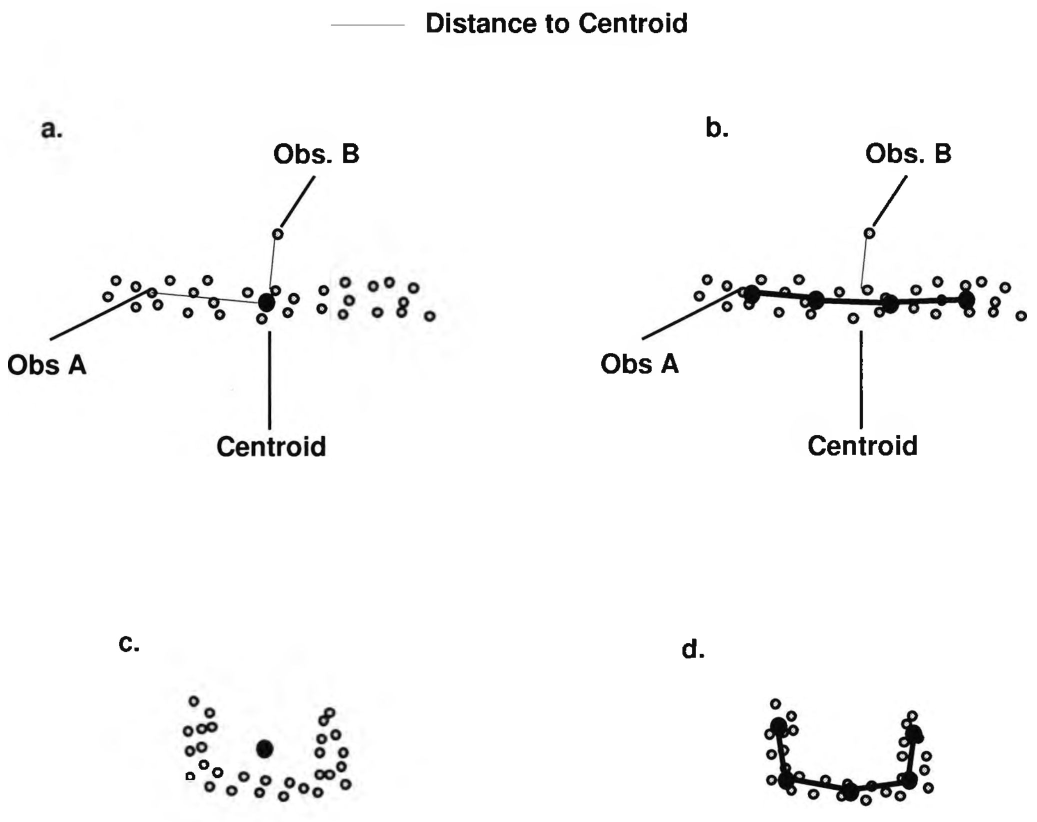


Figure 1.2: The use of the linked line segment representation (*b* and *d*) allows for more accurate measurements of how individual patterns reside within a cluster compared to using the centroid of the cluster (*a* and *c*).

For example, given a significantly nonhyperspherical cluster, representing the cluster structure with only its centroid can lead to inaccurate measurements of the relative degree of membership of individual patterns if relative membership is measured in terms of the distance of a pattern from a cluster. This is illustrated in Figure 1.2*a* and *b* which show an elongated cluster structure. If one were to use the distances of observations *A* and *B* from the cluster centroid to determine which object was closer to the cluster (and therefore implying that its degree of membership in that cluster is greater), observation *B* which is in a low density area would be considered as being more typical of the cluster even though observation *A* is in a higher density portion of the cluster.

The linked line segment representation helps overcome this problem. In this representation of cluster structure, the distance of the observation from cluster is measured as the distance of the observation from the closest line segment in the cluster. Using this scheme, the relative membership of both *A* and *B* can be measured more accurately (Figure 1.2*b*). The problem of using the centroids to measure cluster membership is further highlighted by Figure 1.2*c* and *d*. Here the cluster has a "U" like shape. The centroid of this cluster lies in the empty, low density, space partially surrounded by the cluster. Observations found in the inner part of the cluster will be treated as being closer to the cluster then patterns lying on the outer edge. Given that the inner and outer edges lie in equally dense portions of the cluster, this makes little sense. Using the linked line segment representation allows the inner and outer portions of the cluster to be treated equally.

1.4 Objectives

The objectives of this thesis centre on establishing the need for the APC clustering methodology in data analysis, discussion of its functionality and implementation, and the evaluation its performance.

The demonstration of the need for a clustering methodology like APC involves the evaluation of clustering algorithms in regards to their applicability to large data sets, ability to perform robustly under noisy conditions in addition to their ability to extract nonhyperellipsoidal cluster structures. It will be argued (Chapter 3) that few clustering methods exist that can be of practical use when all three of the above conditions exist together.

The second objective is to introduce the APC approach to data clustering. In Chapter 4 the APC methodology is explained in detail while different strategies that could be taken in the implementation of each stage of the APC methodology are evaluated. For example, as APC relies on an initial clustering of the data, a number of different pattern partitioning methods for use in the in the initial clustering stage are examined and empirically compared (Chapter 6).

The final objective is to evaluate APC in terms of its ability to perform under noisy conditions and recover nonhyperellipsoidal clusters. This is accomplished over a wide range of data sets using Monte Carlo simulation in Chapter 7. A number of different factors are examined such as cluster shape, dispersion, noise, dimensionality and cluster proximity. The objective is to demonstrate that APC is able to perform well under these conditions relative to other hybrid approaches to clustering that could also be applied to large data sets. In addition, the linked line segment approach to cluster representation is evaluated in Chapter 8 using cluster structures that cause principal components analysis and centroid based representations to fail or over parameterise the relevant characteristics of the data. This is done through an empirical comparison of how well these methods recover salient structures known to exist in artificial data via series of data reduction exercises. The use of APC as a data reduction method on two real world data sets is also included.

1.5 Original Contributions

There are three primary original contributions of this thesis to the clustering literature. The first is the introduction and evaluation of an efficient, robust and flexible clustering methodology designed for exploring large and complex data sets. The two stage cluster recovery process of APC is essentially a modification of the hybrid clustering method (WHM) developed by Wong (1982) which also measures intercluster distance as being proportional to intercluster density. However, this thesis introduces two new density based distance metrics which enable APC to recover cluster structure more robustly under noisy conditions than WHM. Although this performance enhancement is paid for by a higher computational cost and some additional parameters, APC is still applicable to large data sets without too great of an additional computational effort. APC also differs from WHM in the manner in which clusters are agglomerated.

The second original contribution of this thesis is the use of connected line segments for representing cluster shape. The application of this technique to feature extraction and data reduction is also examined where it is demonstrated that it can give a more accurate representation of nonlinear feature shapes than k-means or principal components analysis when these features correspond to high density data clusters.

Finally, a number of clustering methods are evaluated in terms of their ability to recover nonhyperellipsoidal cluster shapes. As APC relies on an initial pattern partitioning, Monte Carlo simulations comparing k-means, moving methods and two unsupervised neural networks are examined using a wide variety of cluster structures. In general, previous comparisons of these pattern partitioning algorithms utilised mixtures of multivariate normal distributions. The experiments performed here give a much broader understanding of these algorithms' relative performance and empirically demonstrates the effects of cluster shape on the relative performances of these clustering methods.

1.6 Scope of Thesis

Most clustering methods are heuristic approaches to data exploration. As APC has been developed in very much this same tradition, no attempt is made to place APC into a theoretical context. It is not derived from any preconceived model or theory regarding clustering analysis beyond the goal of developing a method that can perform under the conditions described above. Instead, it can be thought of as a collection of heuristic techniques for recovering cluster structure with flexibility and efficiency. The algorithmic components that make up the APC methodology are used because they are practical approaches to accomplishing the tasks particular to that stage of the methodology.

The main purpose of the literature review (Chapters 2 and 3) is to argue that there is a dearth of clustering techniques available that can be applied to large, noisy data sets with nonhyperellipsoidal cluster shapes. Given the vast scope that can be found in the clustering literature, this discussion is largely limited to examining commonly used approaches to clustering in addition to discussing a number of additional approaches that were influential to the development of, or share characteristics with, APC. Also, this thesis is concerned with the hard clustering of numerically coded data. Therefore, other areas of cluster analysis such as fuzzy clustering (*e.g.* Hathaway and Bezdek 1995; Bezdek, 1981), conceptual clustering (*e.g.* Ralambondrainy, 1995) and image analysis (*e.g.* Jawahar *et al*, 1995; Brunelli, 1992) are not discussed.

The empirical sections evaluating APC are meant to illustrate APC's applicability to noisy data sets with nonhyperellipsoidal clusters. However, the computational costs of Monte Carlo simulation limits the range of clustering algorithms that can be empirically evaluated along with APC. Comparisons of clustering ability (the ability to recover the correct, *a priori* known cluster structure) is restricted to other hybrid pattern partitioning/hierarchical clustering methods as they are amongst the more efficient clustering methods in use. Also, as APC is a modification of a well established hybrid approach to clustering, it is important that APC be evaluated in comparison to a range of these types of clustering methods even if this is at the expense of making comparisons with other clustering approaches. Nonetheless, even though the empirical comparisons have been limited to other similarly hybridised approaches to clustering, these experiments do demonstrate the ability of APC to perform under the conditions for which it is designed.

Finally, the demonstration of APC's linked line segment approach to cluster representation is meant to be largely illustrative. Chapter 8 is meant to show the effectiveness of the use of this type of cluster representation in measuring the cluster membership of individual observations where there are highly nonhyperellipsoidal cluster structures corresponding to high density clusters in the data. This is accomplished through the use of APC in a data reduction exercise. However, this thesis is not intended to be an thorough examination of data reduction techniques.

1.7 Thesis Overview

The remainder of this thesis is structured as follows: Chapter 2 is a discussion of the definition of a cluster used by APC and other clustering methods. The purpose of this chapter is to make clear the definition of a cluster used in APC and in the empirical sections of this thesis. Chapter 3 is a brief review of the cluster analysis literature which discusses the relative advantages and pitfalls of a number of the more commonly used clustering techniques. This chapter also covers the various clustering algorithms used in the empirical portion of this work as well as a number of other methods related to or incorporated into the APC methodology. Chapter 4 is a detailed introduction to the APC methodology. Each of the different data processing stages of APC are examined and various implementation strategies are discussed.

Chapter 5 outlines the Monte Carlo methodology used to evaluate partitioning techniques in Chapter 6 and APC in Chapter 7. In addition, the types of data sets used for experiments in Chapters 6 - 8 are introduced. Chapter 6 takes a closer look at the first stage of APC - the extraction of an initial set of clusters prior to agglomeration.

Four clustering techniques that could be used at this stage are examined: k-means, the moving method and two unsupervised neural networks. The purpose of the chapter is to determine the pros and cons of these pattern partitioning algorithms for use in the initial clustering stage of APC in addition to exploring how cluster shape can effect the relative performances of these clustering algorithms. The results indicate that "winner take all" unsupervised networks, k-means and the moving methods *on average* perform equally well in terms of cluster recovery under a variety of conditions. However, it is also shown that different algorithms perform better than others on different cluster shapes. Moreover, as the moving method is found to converge significantly faster than k-means and does not require the relatively large number of iterations through the data set needed by unsupervised neural networks, it is argued that the moving method is perhaps the best of the pattern partitioning methods examined for use with large data samples. The simulations in this chapter differ from previous comparison studies involving these algorithms in that a wide variety of cluster shapes were examined while most previous Monte-Carlo evaluations used mixtures of multivariate normal clusters (*e.g.* Mangiameli *et al*, 1996; Balakrishnan *et al*, 1996, 1994; Chen *et al*, 1995; Cowgill, 1993; Milligan *et al*, 1983; Milligan, 1981b, 1980; Bayne *et al*, 1980; Milligan and Isaac, 1980; Blashfield, 1976; Kuiper and Fisher, 1975). In addition, all three of these methods have never been simultaneously evaluated on the same data sets.

Chapter 7 consists of a comparison study of APC and other hybrid clustering techniques. In short, an initial clustering of the data is found via the moving method. Using this initial clustering as input, APC is compared via Monte Carlo simulation to WHM and a number of other hybrid hierarchical techniques as a method of agglomerating the initial clusters to recover the known cluster structure of the data. A variety of cluster shape structures are used in addition to examining the effects of cluster dispersion, dimension, noise and initial number of partitions. The results demonstrate that on average (taken over all conditions), APC performs as well as or better than all other methods examined in terms of its ability to correctly recover the known cluster structure in the data.

In Chapter 8 the use of linked line segments for cluster representation is discussed. APC's use of linked line segments as a data reduction method is compared to a centroid based representation of data features and principal component analysis. A number of real and simulated data sets are pre-processed using APC, principal component analysis and k-means and then used an input to a linear discriminant analysis model (LDA). The performance of the pre-processing techniques are then compared based on the error performance of the LDA using the transformed data. The results of this experiment show that the use of the linked line segment representation of cluster structure introduced here leads to an improved representation of data features over that of PCA or centroids on the data structures examined. Finally, Chapter 9 consist of some concluding remarks and suggested directions for future research.

2. APC and the Clustering Problem

The purpose of this chapter is to define the term “cluster” and to discuss cluster analysis in terms of the types of cluster structure that can be extracted. The reason for beginning with this discussion is that before reviewing clustering techniques in general (Chap. 3) it is important to understand what is meant by the term "cluster". As different algorithms use different definitions of a cluster, one needs to know what types of cluster structure particular clustering techniques are capable of finding in the data. This chapter begins with a discussion of why it is important to understand the underlying definition of a cluster that individual clustering techniques operate on. Next, a survey of the various definitions given to the concept of a "cluster" is given. Finally, the chapter closes with a examination of the definition of a cluster underpinning the APC methodology.

2.1 Introduction

When utilising clustering techniques it is important to understand what types of cluster structure the clustering methods one wants to employ are capable of finding. Different clustering algorithms are generally designed to find either specific types of cluster structures or are in other ways limited in the scope of structure they can extract. Because specific clustering techniques will attempt to discover structures in the data that correspond to the underlying definition of a cluster that particular technique operates on, the analyst must choose carefully which clustering approaches to use in order to help ensure that no potentially useful perspectives on the data are overlooked (Balasubramaniam *et al,* 1990). For example, if one uses Ward's method or k-means to analyse a data set, one may fail to detect cluster structures in the data that are nonhyperellipsoidal in structure as both of these methods implicitly define clusters as hyperellipsoids. Even if one has reason to believe that only hyperellipsoidal clusters occur in the data, it is still good practice to apply a number of different methods to help confirm the structures found using one method by applying another (Hand, 1981).

The primary purpose of this chapter is to give some discussion regarding what is meant by the term "cluster" and how different definitions of clusters are incorporated into APC and other clustering techniques. Various definitions of clusters that can be found in the cluster analysis literature are examined and the relationships between a number clustering algorithms and their underlying definitions of a cluster are discussed. In addition, this chapter is also intended to explain more thoroughly the definition of clusters as areas of relatively high density as used by APC. For the purposes of this work, an area of relatively high density is meant to be understood in the sense of *natural* clusters as described in Hartigan (1975). The advantage of the natural definition of a cluster is that there are no inherent shape constraints. The discussion of APC's operating definition of a cluster is also intended to help give an understanding of where APC fits within the range of clustering techniques. It is the natural definition of a cluster that APC is designed to recover and it is with this definition of a cluster in mind that the performance of APC will be compared with other clustering algorithms.

2.2 Finding Clusters Requires Defining Clusters

The definition of a cluster is a fundamental question in cluster analysis which unfortunately has no clear cut answer in the general sense and can even pose difficulties when confined to specific problem areas (Everitt, 1981). On a practical level problems can arise when one notes that different clustering techniques are designed to extract different types of structure from the data. Because of this, the way one defines clusters determines how one searches for clusters and the way in which one searches for clusters determines what kinds of clusters one can find (Anderberg, 1973; Gordon, 1981). The types of structure particular algorithms extract from data depends on the underlying definition of a cluster that the algorithm operates on. In fact, as suggested by Blashfield (1976) one can create a taxonomy of clustering algorithms based simply on how individual clustering algorithms define clusters.

The analyst must understand the relationship between clustering algorithms and their underlying assumptions about cluster structure or risk naïvely allowing the clustering algorithms define what a cluster is for them. Therefore, one must not only understand how individual clustering algorithms define clusters but also how this definition translates into their particular problem domain before one engages in cluster analysis. For example, if one suspects that the clusters in the data may have nonhyperellipsoidal cluster structures due to nonlinear relationships between variables, single linkage may be a good choice of clustering technique. Alternatively, if one is has reason to believe that whatever cluster structures exist in the data will be roughly hyperellipsoidal in structure, one could use k-means. However, although k-means has been shown to be one of the better methods of partitioning data (*e.g.* Balakrishnan *et al*, 1994, 1996; Bayne *et al*, 1980) it tends to find clusters of relatively spherical, compact shape (Blashfield, 1976; Dubes and Jain, 1976; Gordon, 1981). This could lead to misrepresentations of the structures in the data if the structures in the data are not hyperellipsoidal in shape but rather more elongated or "chaining". If one is looking for particular types of structure in the data, one must be sure the clustering algorithms one is using is even capable of finding those structures. Most often however, cluster analysis is a data exploratory exercise where one has little or no *a priori* idea what the cluster structures are like. Because of this it is good practice to apply a number of methods that can extract different types of cluster structure.

Bearing this in mind it is important to make clear what types of cluster structure APC is intended to detect in data before going into too much detail about the APC methodology. Once this is understood, the role APC has to play in cluster analysis and its performance compared to other clustering techniques can be appreciated.

2.3 What is a Cluster?

A number of definitions of the term "cluster" will now be examined, initially following a discussion given by Everitt (1981). Following this, more detailed attention will be paid to the view of clusters as areas of relatively high density. As mentioned above, formally giving a definition to the concept of a cluster is difficult if not potentially misleading. In practice, the precise definition of a cluster depends on the nature of the particular problem at hand and perhaps can only have meaning within a particular context as different definitions of a cluster may have different degrees of relevance to particular problem domains (Bonner, 1964; Hand, 1981).

2.3.1 General Definitions of a Cluster

For a dictionary definition of a cluster one could turn to *A Dictionary of Statistical Terms* where a cluster is defined as

> "A group of contiguous elements of a statistical population; for example, ... a consecutive run of observations in an ordered series or a set of adjacent plots in one part of a field." (Kendall and Buckland, 1982).

This is a fairly high level definition that amounts to stating that clusters are "groups" of data, events or observations. This is the type of general definition that intuitively comes to mind when asked what a cluster is: a cluster is set of observations that are in some way similar.

Wallace and Boulton (1968) also attempt a generalised definition of a cluster defining it as a subset of entities that can be treated as functionally equivalent within some specific context. This definition follows from Bonner (1964) and Hand (1981) who point out that a cluster can only have meaning within a contextual framework. Due to the difficulty in attempting to define what exactly is a cluster and the fact that what is a cluster in one context may not necessarily translate into another, the definition of a cluster may only really have meaning within a particular application.

2.3.2 Computational Contexts

Others have emphasised placing the definition of a cluster within a computational context from which one can construct algorithms to detect the clusters in data. Gengerelli (1963), for example, defined a cluster as an aggregate of observations where the distance between two observations within a cluster is less that the distance between any observation in the cluster and any observation outside the cluster. The single linkage clustering algorithm and related techniques are a direct manifestation of viewing clusters this way (Blashfield, 1976). Blashfield also points out how minimum variance methods such as Ward's method, k-means and ISODATA (Ball and Hall, 1965) implicitly define clusters as groups of observations where the sum squared distance of the observations within the clusters to their corresponding centroids are minimised. Effectively, clusters are defined as compact groupings of observations.

2.3.3 Cohesion and Isolation

Cormack (1971), Jardine and Sibson (1971) and Gordon (1981) refer to concepts of internal cohesion and external isolation when attempting to define what a cluster is. Internal cohesion refers to objects within a cluster sharing some degree of similarity while external isolation refers to the degree of "remoteness" a given grouping of observations has from the rest of the observations. Figure 2.1 gives some examples of clusters showing cohesion and isolation. Isolation means that these groupings of alike objects are away and apart from other groupings. All of the clusters in Figure 2.1 are isolated in this sense. Cohesion on the other hand refers to similar or "alike" observations are found together. In the case of the clusters in Figure 2.1 cohesion can be taken as meaning similar observations coexist in a similar portion of the pattern space. This definition however, can break down in situations as in Figure 2.2 displaying a dumbbell shaped structure. As Gordon (1981) points out, the intermediate observations between isolated groupings of observations prevent the construction of truly isolated clusters unless the entire structure is treated as a single cluster.

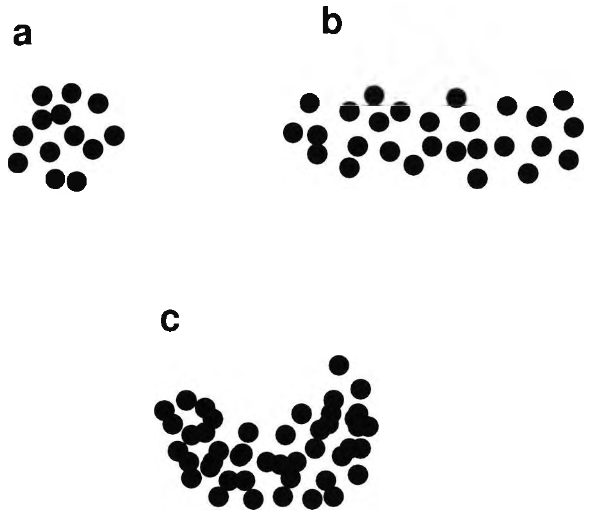


Figure 2.1: Clusters exhibiting cohesion and isolation.

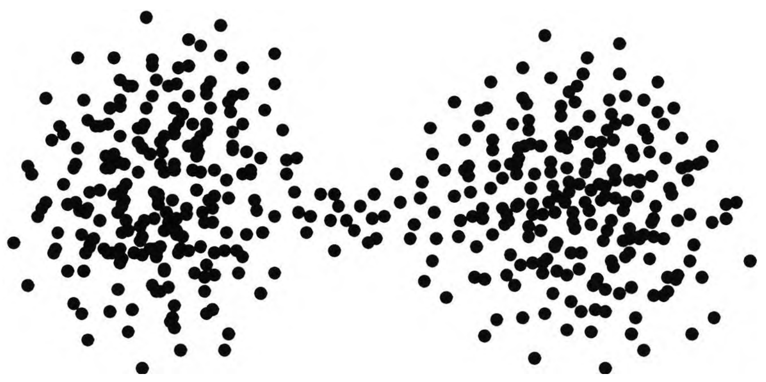


Figure 2.2: Statements of cohesion and isolation are difficult to make when there are linking areas between clusters.

2.3.4 Chained and Compact Clusters

Rice and Lorr (1969) and Johnson (1967) make a distinction between chained and compact clusters. Chained clusters are clusters where each member of a given cluster is more similar to at least one other member of the cluster than it is similar to any member of any other cluster as in Gengerelli's definition. In compact clusters all members of a given cluster are more like *every* other member of the cluster than they are to any other member of any other cluster. Complete linkage, where the distance between clusters is measured based on the distance between the furthest pair of objects, one from each cluster, is a clustering algorithm explicitly designed to find these types of compact clusters. Cattell and Coulter (1966) produce a similar dichotomy: segregates and homostats (Lorr, 1983). In Cattell and Coulter's terms a segregate is a set of "entities continuously related through other entities in the cluster and isolated from others outside the cluster but not necessarily similar in position". In other words, the objects are similar on a continuous basis but objects at one the end of the continuum may not be very similar to objects at the other extreme. As Lorr (1983) points out, this can be viewed as a chained cluster. A homostat is a set of entities occupying a similar area of the pattern space - a compact cluster if you will.

2.3.5 Natural Clusters

Assuming a geometrical conception of entities or observations existing in *d*-dimensional space (*e.g.* Euclidean space) - clusters have been defined as areas of relatively high density surrounded by areas of relatively low density (Backer, 1995; Everitt, 1993,1981; Lorr, 1983; Hartigan, 1975). Hartigan (1975) refers to these as *natural clusters* and formally defines a natural cluster as follows: Let $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, ...,$ $, \mathbf{x}_n\}$ a set of objects in a *d*-dimensional space. $f(\mathbf{x}_n)$ is a value proportional to the density of points at a fixed volume centred at some given object, $\mathbf{x}_n$. A cluster is defined as a connected subset[7], C_k, of $\mathbf{X}$ where each object in the subset has a value $f(\mathbf{x}_n)$ greater than or equal to some threshold f_t. In other words, an area of relatively high density is a connected subset of patterns where $f(\mathbf{x}_n) \geq f_t$ for all $\mathbf{x}_n \in C_k$.

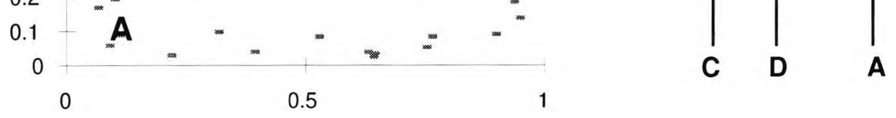


Figure 2.3: A density contour tree created from this data set. With $f_t = 0$, all of the clusters are treated as a single cluster. As f_t increases, the different clusters begin to separate. Initially, cluster A (the surrounding low density area) is separated from cluster B, the high density square. Increasing f_t further, causes clusters C and D to be separated from cluster B. The density based relationships between clusters can be represented with the hierarchical tree (dendrogram - see Chapter 3) on the right.

[7]A connected set is a set C_k of observations where any two observations are linked by a *cycle* whose members are also in the set C_k. A cycle between two observations $\mathbf{x}_i$ and $\mathbf{x}_j$ is a sequence or *path* of p observations $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_p$ where each observation $\mathbf{x}_m$ is linked to $\mathbf{x}_{m+1}$ and $\mathbf{x}_1 = \mathbf{x}_i$ and $\mathbf{x}_j = \mathbf{x}_p$. See Hartigan (1975).

Natural clusters can also be easily incorporated into a hierarchical clustering strategy. Hartigan (1975) shows that by varying the value of f_t one can generate a hierarchical *density contour tree* of **X**. At $f_t = 0$ the entire data set is modelled as a single cluster as every observation in the data set has a density greater than 0. Systematically increasing f_t will cause the data to be divided into relatively high and low density areas. For example, after increasing f_t the roughly square area in the centre of Figure 2.3 separates from the surrounding area as it is of higher density. Within this cluster there are also two other clusters of relative high density which would become separated at higher values of f_t. Regardless of how one wants to treat the different levels of the hierarchy, each level defines a cluster partitioning based on that level's corresponding f_t value.

2.4 Relative Density as a General Definition of a Cluster

As suggested above, different definitions of a cluster can apply to the same structure. For example, a chained cluster can also be defined as an area of relatively high density. Figure 2.4 displays a chaining cluster that would certainly fit the description of a segregate or chaining cluster as the observations are related to each other on a continuous basis. Alternatively, one could also view Figure 2.4 as a cluster which fits the definition of being an area of relatively high density. Observations within the structure occur at a higher density than do observations in other parts of the pattern space. An algorithm using a density search approach could therefore also be applied to find this type of cluster.

The natural definition of a cluster can also be applied to the conceptual dichotomy of compact and chained clusters. Arguably, the distinction between compact and chained clusters is really no different from stating that there are two types of clusters: (1) compact, hyperellipsoidal clusters and (2) noncompact, nonhyperellipsoidal clusters. Defining a cluster as an area of relative high density encompasses both definitions as both definitions can be seen as corresponding to areas of relatively high density. The concepts of cohesion and isolation can also be applied to natural clusters. Highly dense areas of the pattern space are by definition cohesive while isolation in terms of natural clusters translates into stating that high density clusters are separated from other clusters by low density areas in the pattern space.

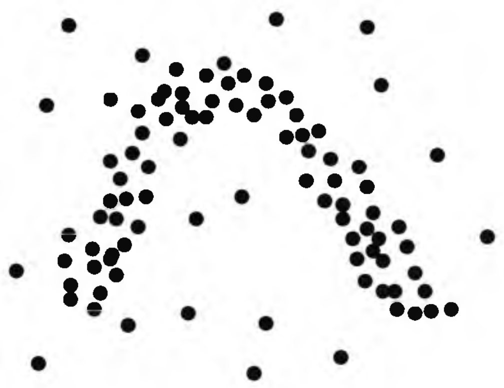


Figure 2.4: An example of a chaining cluster.

At this point, one could argue that clustering algorithms that use a chaining definition of a cluster can also extract hyperellipsoidal clusters as these structures are not inherently ruled out by the chaining definition of a cluster (the opposite cannot be said of algorithms using hyperellipsoidal definitions of clusters). However, defining clusters as continuously related observations or as areas of high density can have an important impact on the performance of algorithms designed to detect such structures. As will be discussed in later chapters, algorithms such as single linkage and MST that utilise cluster definitions based on continuous relationships between observations with out reference to relative density tend to perform poorly under noisy conditions, particularly when clusters are not well separated. The reason for this is that transitive relationships between observations sought by these techniques can cause these methods to "get lost" amongst the noise and lose track of the underlying cluster structure. The "point density" approach to defining natural clusters taken in this work and incorporated into other density seeking methods helps overcome this problem by taking into account local density.

By basing a clustering method on the natural definition of a cluster one, theoretically at least, can get around some of the problems inherent in other clustering techniques. For example, k-means cannot extract chaining clusters because it implicitly defines clusters as being compact. Areas of relatively high density, on the other hand, are not inherently restricted to particular shapes clusters (Everitt, 1981). Although single linkage an extract both compact clusters and chaining clusters (*i.e.* arbitrarily shaped clusters), it tends to erroneously agglomerate clusters that are in close proximity or separated by noise due to its definition of clusters as being composed of objects continuously related to each other. Similar problems occur with MST based approaches that also use the chaining definition of a cluster. By building a clustering technique around a definition of a cluster as an area of relatively high density, one can both model compact and chaining clusters while reducing the likelihood of false agglomerations of clusters residing in close proximity or in noisy portions of the pattern space as the clusters should still be separated by low density areas.

As pointed out by Openshaw (1995), it is important for a clustering algorithm to not be too restricted in terms of the types of cluster structure they can extract. The advantage of a clustering algorithm designed around clusters defined as areas of high density is that there is less likelihood of structure being overlooked in the data as cluster shape is not implicit in the algorithm's underlying definition of a cluster.

2.4.1 Problems with Natural Clusters

Nonetheless, the natural definition of clusters can still lead to problems. For example, Figure 2.2 can be viewed as a single dumbbell shaped cluster or two or more clusters. Using Hartigan's notion of natural cluster, which view one takes could be influenced by the value one gives to the radius, r, of the unit volume used to calculate f_t. If r is relatively large, then this figure would be modelled as three clusters in a hierarchy. However, if r is relatively small, the whole figure would be classified as a single cluster because within the dumbbell shape, the patterns are distributed uniformly. Strictly speaking, as the dumbbell is a connected region of relatively high density, it should be treated as a single cluster. However, one must first calculate this density and whether or not Figure 2.2 is actually treated as one or three clusters depends on how one calculates this density estimate. This highlights an important point about using the natural definition of a cluster. Although defining clusters as natural clusters allows the possibility of modelling arbitrarily shaped clusters, it does not in itself guarantee that one will be able to find such clusters. Success or failure to recover particular cluster shapes in a data set depends in large part how one incorporates the natural definition of a cluster into the clustering algorithm.

2.4.2 Clusters of Arbitrary Shape

Before leaving the subject of defining the concept of a cluster, the term *arbitrary shape* in reference to cluster structure should be clarified. Clusters can come in any shape or form. Figure 2.5 is an attempt to convey the infinite and arbitrary shapes clusters can have. Clusters can be compact, elongated, have variable point densities, have hierarchical relationships between each other and so on. There are no limits to the forms or shapes clusters can take (see Osbourn and Martinez (1995), Backer

(1995) and Fromm and Northouse (1976) for further discussions of cluster shape). What is important, though, is that a clustering algorithm applied to a given data set is capable of modelling clusters that might be present. In other words, clustering algorithms should have sufficient scope to handle a wide range of cluster shapes. If a given algorithm can only model clusters of compact shape, its use in data sets that contain clusters like many of those in Figure 2.5 could lead to a failure to detect important structures present in the data.

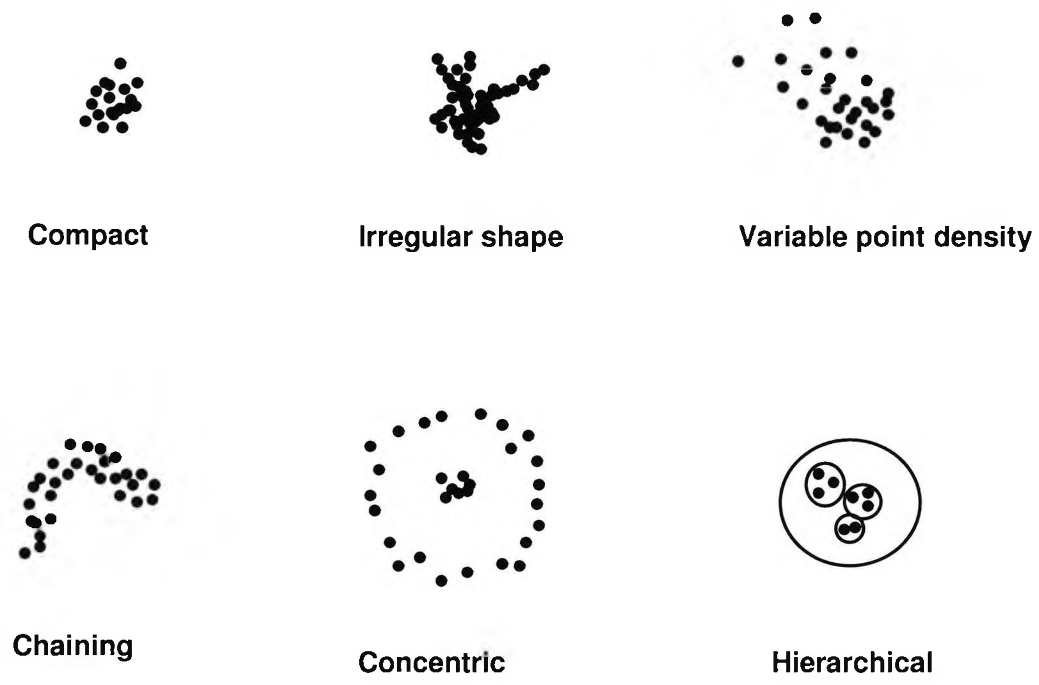


Figure 2.5: Various cluster shapes.

2.5 The APC Definition of a Cluster

This thesis can be viewed as *the development and evaluation of a clustering methodology intended to detect and model clusters defined as areas of relatively high density*. In short, the APC methodology can viewed a way of detecting and representing areas of relatively high density of arbitrary shape that are piecewise linear approximatable. In other words, cluster shapes that can be modelled or fit by a string of connected line segments. This includes not only identifying clusters in the pattern space, but also the capacity to take into account the *shape* of the cluster when assessing information about the cluster memberships of individual observations. The initial pattern partitioning and intercluster density search stages of APC together comprise the cluster detection and identification aspect of the APC approach to the clustering problem. The use of line segments to determine cluster membership of individual objects is an attempt incorporate cluster shape when taking measurements regarding where an object resides within a cluster.

The above discussion is intended as a justification for the natural definition of a cluster in APC and as a guide to where within the range of clustering techniques APC belongs. Because the natural definition places no restrictions on shape, clustering algorithms using this definition are more likely to be able to recover nonhyperellipsoidal cluster shapes. APC can therefore be thought of as a method of implementing the natural definition of a cluster into a clustering methodology intended for the recovery of arbitrarily shaped clusters.

2.6 Summary

As different clustering methods arise from different definitions of a cluster, results one obtains in any given situation depends in large part on the *a priori* definition of a cluster the particular clustering algorithm used. It is therefore important that the analyst understands the definition of a cluster that each clustering algorithm operates on.

The operating definition of a cluster for APC is an area of high density surrounded by an area of relatively lower density. This definition has the benefit of being intuitively appealing while not in itself restricting the types of cluster shape a clustering method based on this definition can recover. APC is an attempt to incorporate this definition of a cluster into a clustering methodology that preserves the generality and inherent flexibility of the natural definition of a cluster. As long as this definition is relevant to one's problem domain, APC may be an appropriate clustering methodology to consider.

3. Overview of Clustering Algorithms

While the previous chapter discussed the relationship between clustering algorithms and their underlying definition of a cluster, this chapter is primarily concerned with the clustering algorithms themselves. The purpose of this chapter is two fold. First, the need for a clustering methodology like APC is demonstrated by arguing that there is a dearth of clustering methods that can be applied to large, noisy data sets containing arbitrarily shaped clusters. A range of clustering methods are evaluated in term of their applicability to data sets possessing these attributes. The other objective of this chapter is to discuss approaches to clustering that can be incorporated into or that influenced the development of APC.

3.1 Introduction

The clustering techniques discussed in this chapter will be compared and contrasted in terms of computational efficiency, sensitivity to noise and flexibility in regards to the types of cluster structure they can extract. This survey is not meant to be exhaustive - the range of clustering algorithms is quite large (Backer, 1995) and so it is beyond the scope of this work to review everything that has been produced. Rather, this survey is primarily concerned with general approaches to hard clustering in order to demonstrate the lack of clustering algorithms that can adequately cope under the conditions APC is designed to operate. There is also an emphasis on clustering techniques that can be incorporated into the APC methodology at the initial pattern partitioning, intercluster density estimation or the agglomeration stages. Finally, some additional algorithms and approaches to clustering are also discussed because they represent techniques which influenced the development APC.

The remainder of this chapter is divided into four sections: comparing clustering methods, hierarchical clustering methods, partitional methods and hybrid methods. The first section discusses criteria that can be used to compare clustering techniques and establishes the perspective through which the various clustering methods discussed in this chapter are to be evaluated. The next two sections review a number of hierarchical and pattern partitioning clustering algorithms. Finally, a number of hybrid approaches will be examined that incorporate both hierarchical and partitional clustering. Like these other hybrid approaches, APC is an attempt to bring together many of the most appealing aspects of a number of clustering methods to recover clusters quickly, efficiently and with flexibility. Moreover, it is in comparison with these hybrid methods that the clustering abilities of APC will be empirically evaluated (Chapter 7).

3.2 Comparing Clustering Algorithms

As pointed out by Dubes and Jain (1976) it is difficult to make general statements regarding superiority of one clustering algorithm or another. In their comparison paper, they rely on two general approaches. The first is to compare the performance of different clustering methods on the same data sets to determine their relative abilities to recover known cluster structures. This approach is taken in Chapters 6 - 7 The second method is to compare clustering techniques based on a set of *admissibility criteria* (Fisher and Van Ness, 1971; Rubin, 1967). In short, an admissibility criterion is a property that should be possessed by any "reasonably" useful clustering technique. For example, Fisher and Van Ness discuss properties such as the ability to find well separated clusters and the robustness of the clustering solution when patterns have been duplicated. Given that this thesis is proposing a clustering method applicable to large, noisy data sets with arbitrarily cluster shapes, four admissibility criteria for determining "usefulness" under these conditions will be used in evaluating clustering methods in this chapter: (1) CPU costs, (2) memory costs, (3) flexibility in regards to the types of cluster shape clusters that can be recovered and (4) robustness under noisy conditions.

3.2.1 Computational Efficiency

Due to the decreasing costs of storing data it is important that clustering algorithms exists that are sufficiently computationally efficient to be practical for use on large data sets. Similarly, algorithms that are used to process these data sets must not impose additional large storage requirements in order to accomplish the analysis. For example, many implementations of hierarchical techniques require a matrix to be calculated and stored consisting of every interobservation distance. Obviously, this is not practical for very large data sets.

3.2.2 Cluster Shapes

As suggested by Openshaw (1995) clustering algorithms should not be too restricted in the types of cluster structure they can detect. Many situations exist that involve data sets that have no restrictions on the structure of clusters. For example, geospatial analysis may require the modelling of geographical areas with no inherent constraints on the shape of the areas of interest.

3.2.3 Noisy Data

Another important requirement is robust performance under noisy conditions. Real world data is often extremely noisy. Noise can arise form a variety of sources -

measurement error, recording errors, missing data, sampling errors and the unavoidable inclusion of information in the data that has no relevance to the analysis at hand. It is important, therefore, that clustering techniques applied to data that may contain moderate to high levels of noise be able to cope with these conditions.

Whether a given data set will require all of these characteristics in a clustering technique depends on the problem domain on hand. For example, if the clusters in the data are all multivariate normal in nature, ability to model clusters of any other form is not necessary. If the size of the data set is small, computational efficiency and memory usage are not important issues. In any case, no algorithm exists that can perform under all conditions at a satisfactory level on any or all problems. Often good performance in one characteristic is bought at the cost of poor performance on another characteristic. For example, k-means is computationally fast but tends to only find clusters of roughly hyperspherical shape. Single linkage can extract arbitrary cluster shapes but is both computationally expensive and sensitive to noise in the data. When comparing the relative abilities of clustering algorithms one must recognise this fact, and perhaps only talk about various algorithms as being better at particular problems or under particular conditions.

3.3 Hierarchical Clustering Techniques

Hierarchical techniques are characterised by the way they generate nested, hierarchical relationships between clusters. The hierarchy is such that at the bottom all individual observations are treated as individual clusters. At the top of the hierarchy, the entire data set is treated as a single cluster. Hierarchical clustering techniques themselves can be categorised into agglomerative and divisive algorithms, depending on how the hierarchy is created. Agglomerative hierarchical clustering techniques begin by treating each observation as an individual cluster. The two closest observations or clusters are then found and agglomerated into a new cluster.

This process is then repeated until the entire data set is agglomerated into a single cluster. Although agglomerative hierarchical clustering algorithms generally follow this same basic procedure, they differ primarily in how the distance between clusters is calculated. The general form of agglomerative hierarchical clustering algorithms is as follows:

Let **X** be a set of N, d-dimensional observations in Euclidean space, $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\}$, where $\mathbf{x}_N = \{x_{N1}, x_{N2}, ..., x_{Nd}\}$ and let $\boldsymbol{C}$ be a set of k groups or clusters in the data, where $\boldsymbol{C} = \{C_1, C_2, ..., C_k\}$.

1. Initially treat each observation as an individual cluster:

$$\{C_1, C_2, ..., C_k\} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\} \qquad 3.1$$

$$k = N$$

2. Find the two closest clusters, C_a and C_b such that

$$d^*(C_a, C_b) < d^*(C_i, C_j) \quad \text{for all } i \text{ and } j,\ i \neq j \qquad 3.2$$

where $d^*(C_i, C_j)$ is the distance between clusters C_i and C_j.

3. Create a new cluster C^t_g consisting of the agglomeration of the two closest clusters such that $C^t_g = \{C_a \cup C_b\}$ where C^t_g is the cluster created at the t^{th} agglomeration.

4. Increment t, and repeat steps 2 and 3 using C^t_g in place of C_a and C_b until the entire data set has been agglomerated into a single cluster:

$$C_g^{N-1} = \{\mathbf{x}_1, \mathbf{x}_2\ ..., \mathbf{x}_N\} \qquad 3.3$$

The resulting process of agglomeration is often represented as a *dendrogram* indicating which clusters are agglomerated when (Figure 3.1). The heights of the lines joining different levels of the hierarchy are often drawn proportional to the distance between the clusters agglomerated at that level. This gives the analyst a graphical representation of how the nested cluster structures fit together. As agglomerative hierarchical techniques generally differ in how intercluster distance is calculated (Lance and Williams 1967a), different measures of intercluster distance produce different models of the data.

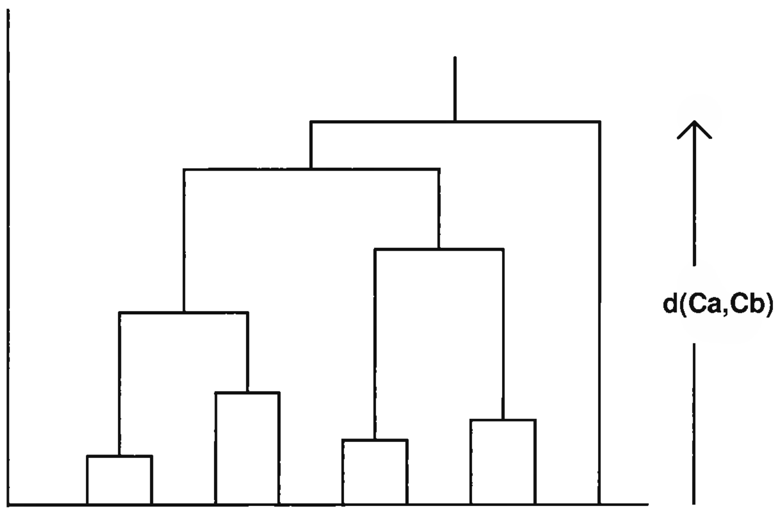


Figure 3.1: Dendrogram produced by hierarchical clustering.

Divisive hierarchical techniques begin by treating the entire data set as a single cluster and then recursively divide the data set up into smaller clusters until each individual observation is treated as a cluster in its own right. As APC agglomerates the initial clustering of the data, divisive methods are not discussed here. The interested reader is referred to Everitt (1993, 1981) or Anderberg (1973) for overviews of these methods.

The following is a brief review of the most commonly used agglomerative hierarchical techniques. The purpose is to both introduce algorithms that will be incorporated into hybrid methods used as comparisons to APC later in this thesis and to highlight some of the advantages and disadvantages of using hierarchical clustering algorithms in general.

3.3.1 Single Linkage

Single Linkage (McQuitty, 1957; Sneath, 1957): The distance between two clusters is defined as being the distance between the two closest observations, one taken from each cluster:

$$d^*\left(C_i, C_j\right) = \min\left[d\left(\mathbf{x}_l^i, \mathbf{x}_m^j\right)\right]$$
$$l = 1,\ 2,\ ....,\ n_i \qquad m = 1,\ 2,\ ...,n_j \tag{3.4}$$

where $\mathbf{x}^k$, C_k and n_k is the number of observations in cluster C_k and $d(\mathbf{x}^i,\mathbf{x}^j)$ is the distance (*e.g.* Euclidean, correlation, *etc.*) between observations $\mathbf{x}^i$ and $\mathbf{x}^j$. Single linkage often produces long chaining clusters which is something of a double edged sword. If a cluster does in fact have a long, chaining or irregular structure, single linkage will be able to extract the cluster quite well. On the other hand, if the clusters are poorly separated or exist in areas of high noise, single linkage will fail to distinguish between the cluster structures (Gordon, 1981).

3.3.2 Complete Linkage

In *complete linkage* (Horn, 1943), the distance between clusters is measured as the distance between the most distant or dissimilar pair of observations, one object taken from each cluster:

$$d^*\left(C_i, C_j\right) = \max\left[d\left(\mathbf{x}_l^i, \mathbf{x}_m^j\right)\right]$$

$$l = 1,\ 2,\ \ldots.,\ n_i$$
$$m = 1,\ 2,\ \ldots, n_j \qquad 3.5$$

The use of eq. (3.5) in complete linkage counters the chaining properties of single linkage and so therefore it has a tendency to find relatively small compact cluster structures.

3.3.3 Average Linkage

Average linkage (Sokal and Michener, 1958) measures intercluster distance as the average distance between each pair wise set of observations between the two clusters:

$$d*\left(C_i, C_j\right) = \frac{1}{n_i n_j} \sum_{l=0}^{n_i} \sum_{m=0}^{n_j} d\left(\mathbf{x}_l^i, \mathbf{x}_m^j\right) \qquad 3.6$$

The idea behind average linkage is that extreme values can be averaged out in measuring intercluster distances. It was also developed to compensate the tendency of complete linkage to produce small compact clusters (Lorr, 1983). Although still generally producing compact clusters, average linkage is capable of recovering clusters of unequal size.

3.3.4 Centroid Linkage

Intercluster distance in *centroid linkage* (Sokal and Michener, 1958) is based on the distances between each cluster's centroid:

$$d*\left(C_i, C_j\right) = d\left(\overline{\mathbf{x}}_l{}^i, \overline{\mathbf{x}}_m{}^j\right)$$

$$\overline{\mathbf{x}}^k = \frac{1}{n_k} \sum_{m=0}^{n_k} \mathbf{x}_m^k \qquad 3.7$$

When two clusters are merged, its distance from the newly agglomerated cluster to other clusters is measured as the distance from the centroid of all the patterns comprising the new cluster to the centroids of the other clusters.

3.3.5 Median Linkage

Median linkage (Gower, 1967) was developed to counter a potential problem with centroid clustering. In centroid clustering, if two groups being agglomerated are of greatly different sizes, the new group centroid will be more similar to the larger group diluting the effects of the smaller group in the formation of the new cluster. Let cluster C_{ij} be the fusion of clusters C_i and C_j. To overcome the excessive influence of the larger cluster, the distance of the centroid of a given cluster C_k to C_{ij} is the length of the median line of the triangle defined by the centroids of C_i , C_j and C_k:

$$d^*\left(C_{ij}, C_k\right) = d\left(\mathbf{c}_k, mp\left[\mathbf{c}_i, \mathbf{c}_j\right]\right) \qquad 3.8$$

where $mp[\mathbf{c}_i,\mathbf{c}_j]$ is the midpoint between the centroids of clusters C_i and C_j. By using the length of the median extending from cluster C_k to the opposite side of the triangle (defined by the line extending between C_i and C_j) rather than the centroid of $C_i \cup C_j$ to measure the distance of C_k from C_{ij}, the clusters C_i and C_j can be treated as being equal in size.

3.3.6 Ward's Method

Ward's method (Ward, 1963) differs from the above techniques in that it incorporates a cost function into the merging process. In short, Ward's method agglomerates clusters that minimise the total within group error at each stage. The total within group error is used as a measure of information loss attributable to each agglomeration operation. At each agglomeration all possible pairs of clusters are examined for agglomeration. The pair of clusters whose agglomeration, C'_g, leads to the lowest increase in information loss are agglomerated. The distance between clusters (*i.e.* information loss) is defined as the mean sum of squares of the objects in the agglomerated cluster:

$$d^*(C_i, C_j) = \frac{1}{n_g} \sum_{m=0}^{n_g} \left(\overline{\mathbf{x}}^g - \mathbf{x}_m^g \right)^2 . \qquad 3.9$$

where $C_g = \{C_i \cup C_j\}$. The smaller the value of eq. (3.9), the smaller the distance between clusters. Ward's method is biased towards equally sized clusters (Blashfield, 1976) and tends to find compact clusters as with complete linkage.

Parameter	α_i	α_j	β	γ
Single Linkage	0.5	0.5	0.0	-0.5
Complete Linkage	0.5	0.5	0.0	0.5
Centroid linkage	$n_i/(n_i + n_j)$	$n_j/(n_i + n_j)$	$-\alpha_i\alpha_j$	0.0
Average linkage	$n_i/(n_i + n_j)$	$n_j/(n_i + n_j)$	0.0	0.0
Median linkage	0.5	0.5	-0.25	0.0
Ward's method	$(n_k + n_i)/(n_i + n_j + n_k)$	$(n_k + n_j)/(n_i + n_j + n_k)$	$-n_k/(n_i + n_j + n_k)$	0.0

Table 3.1: Values for the Lance and Williams (1967a) hierarchical clustering formula.

3.3.7 A Generalised Formula for Hierarchical Clustering

As the above hierarchical techniques all differ in how $d^*(C_i,C_j)$ is calculated, each can be expressed by the general formula:

$$d^*(C_k, C_g) = \alpha_i d^*(C_k, C_i) + \alpha_j d^*(C_k, C_j) + \beta d^*(C_i, C_j) + \gamma \left| d^*(C_k, C_i) - d^*(C_k, C_j) \right| \qquad 3.10$$

where C_g is the agglomeration of C_i and C_j and C_k is any other cluster in the data (Lance and Williams, 1967a). By adjusting the values of α_i, α_j, β and γ formulas for single linkage, complete linkage, centroid linkage, median linkage and Ward's method can be obtained (Table 3.1). Formulating all these methods in this way is useful as it shows how a single algorithm can be used to implement all of these methods. However, using this formulation is not necessarily the best way to implement these clustering methods (Murtagh, 1983).

3.3.8 Minimum Spanning Tree

Graph theoretical methods are widely used in cluster analysis (Backer, 1995). Zahn (1971) initially explored the use of minimum spanning trees in cluster analysis. The Minimum Spanning Tree (MST) approach is introduced here as it is related to single likage. A graph is defined as a connected set, *G*, of points or *nodes*. Relationships between nodes are indicated by the connections between the nodes referred to as *edges*. Each edge has a corresponding *weight* which is a numeric value assigned to the edge. The weight assigned to an edge between two nodes is a quantification of the relationship between the nodes. For example, if the edge weights correspond to geometric distance, nodes connected via edges with small weights lie close together while nodes connected via edges with large weights are relatively far apart. A *path* is defined as the sequence of edges that connect two nodes via other nodes. A graph is said to be *connected* if a path exists between any two nodes and is said to be *complete* if an edge exists between every possible pair of nodes. A *tree* is a connected graph with no cycles (a cycle is a set of connected nodes $V = \{v_1, v_2, \dots, v_n)$ where v_1 and v_n are the same node). A *spanning* tree is a tree consisting of all of the nodes contained in G and the *minimum spanning* tree of G is the spanning tree in which the sum of the weights corresponding to the edges is minimal relative to all other possible spanning trees that could be constructed in G.

Given that removing any one edge from the spanning tree of a data set will result in the partitioning of the data, the basic procedure for finding clusters in data with a MST is as follows:

1. Set $G = \mathbf{X}$ and construct the MST for G where the edge weights correspond to the distance between patterns.
2. Find any inconsistent edges in the MST.
3. Remove the inconsistent edges to partition the data into clusters.

Zahn (1971) discusses a couple of algorithms for finding the MST in step one. One method by Kruskal (1956) is to place the edges of the tree in order from smallest to largest such that each successive edge does not generate a cycle with any previously chosen edge. Another method by Prim (1957) starts with an arbitrary node. The smallest edge connected to this node along with the other node connected to this edge are referred to as fragment tree T_1. Fragment tree T_k is found by adding the node not in T_{k-1} with the smallest edge connected to T_{k-1}. Repeat this process until all nodes have been agglomerated into the tree. Other approaches to constructing minimum spanning trees can be found in Murtagh (1983).

Inconsistent edges are defined as edges that have a significantly larger weight than edges on either side. Zahn recommends determining significance by either looking at how many standard deviations the inconsistent weight is from the average of the weights of either side or by simply looking at the ratio of the inconsistent weight to the average of the weights on either side.

Hartigan (1975) discuss how single linkage clusters can be found via the MST approach to clustering. This can be accomplished by removing links in order of the size of the weights (largest to smallest). The resulting hierarchy is the single linkage clustering of the data. However, removing any one or subset of edges from the MST generates a partition of the data. Depending how one approaches the removal of edges to generate partitions (removing inconsistent edges using the above methods or by first generating a single linkage hierarchy) one can view the MST approach as either a partitioning or a hierarchical clustering method.

3.3.9 Density linkage

Density linkage refers to a class of hierarchical methods that use nonparametric density estimates to measure the distance between observations. Two such methods are the nearest neighbour method and the uniform kernel method (Wong and Lane, 1983). A third method8, called two stage density linkage (Sarle, 1989) is discussed in section 3.5. All three methods are essentially extensions of the single linkage method using density based distance metrics.

3.3.9.1 Nearest neighbour method

The nearest neighbour method (Wong and Lane, 1983), is identical to single linkage in overall structure:

$$d^*(C_i, C_j) = \min\left[d\left(\mathbf{x}_l^i, \mathbf{x}_m^j\right)\right] \qquad 3.11$$

$$l = 1, 2, ..., n_i$$
$$m = 1, 2, ..., n_j.$$

8 Other examples of density linkage can be found in Carmichel *et al* (1968) and Wishart (1969a)).

The difference is in how the distance between individual observations, d($\mathbf{x}^i$,$\mathbf{x}^j$) is calculated where

$$d^*\left(\mathbf{x}^i,\mathbf{x}^j\right)=\frac{1}{2}\left(\frac{1}{f\left(\mathbf{x}^i\right)}+\frac{1}{f\left(\mathbf{x}^j\right)}\right) \quad \text{if } d(\mathbf{x}^i,\mathbf{x}^j)\le \max\left(r_s\left(\mathbf{x}^i\right),r_s\left(\mathbf{x}^j\right)\right)$$

$$=\infty \quad \text{otherwise} \qquad 3.12$$

where $r_s(\mathbf{x}^k)$ is the distance from observation $\mathbf{x}^k$ to its s nearest neighbour and $f(x^k)$ is the proportion of observations within a sphere of radius $r_s(\mathbf{x}^k)$ divided by the volume of the sphere. Effectively this is the single linkage method with the distance between two observations equal to the average of the point density at each of the observations.

3.3.9.2 Uniform kernel method

The uniform kernel method is similar to the nearest neighbour method except that is uses a uniform kernel density estimate (*aka*. naïve density estimate - see Appendix A2):

$$d^{**}\left(\mathbf{x}^i,\mathbf{x}^j\right)=\frac{1}{2}\left(\frac{1}{f\left(\mathbf{x}^i\right)}+\frac{1}{f\left(\mathbf{x}^j\right)}\right) \quad \text{if } d(\mathbf{x}^i,\mathbf{x}^j)\le h$$

$$=\infty \quad \text{otherwise} \qquad 3.13$$

where h refers to the radius of a sphere (*i.e.* bin size) specified by the user and $f(x^k)$ is the proportion of observations within the sphere divided by its volume.

3.3.10 Other Approaches

Other approaches to hierarchical clustering have included the application of statistical mechanics, simulated annealing and other iterative function optimisation approaches to hierarchical clustering (*e.g.* Frigui and Krishnapuram, 1997; Wong, 1993). In general, these methods are similar to Ward's method in that the "optimal" clustering is found at each level of the hierarchy by optimising an objective function. Weighted sum of split and diameter clustering (Wang *et al,* 1996) uses an objective function approach to divisive hierarchical clustering and attempts to balance the effects of searching for both hyperellipsoidal and chaining clusters. A major problem with many function optimisation approaches to hierarchical clustering, however, is that they can be too computationally expensive for use in large data sets.

3.3.11 Criticisms of Hierarchical Techniques

Hierarchical techniques were originally developed for use in the biological sciences where researchers were interested in generating complete hierarchical classifications of their data. Quite often though, hierarchical techniques are applied to data where the appropriateness of applying a hierarchical classification is questionable. For example, if one were interested in classifying psychiatric patients into a number of mutually exclusive groups, generating a hierarchical classification makes little sense. When one is not interested in a complete hierarchical taxonomy of one's data one must determine a suitable cut-off point in the hierarchical tree. The most common method is to look for a point in the dendrogram where the distance between clusters suddenly increases. However, one is not always so fortunate to easily find such a point. Evaluation of methods for determining cut-off points is beyond the scope of this thesis - see Mojena (1977) and Milligan and Cooper (1985) for discussions of other methods of determining cut-off points in the dendrogram. In any case, the use of dendrograms in hierarchical cluster analysis can be a convenient and graphically appealing method for helping to decide on the number of clusters in the data. However, the size and therefore the use of dendrograms becomes overwhelming when one has large numbers of observations.

Another criticism of most of the more commonly used hierarchical methods is that they generally tend to perform best when the clusters are well separated and, with the notable exception of single linkage, relatively compact in shape (Lorr, 1983). Average linkage, complete linkage, Ward's method and centroid linkage are often inadequate when modelling long chaining like cluster as they have a tendency to find hyperspherical shaped clusters. Single linkage is good at finding long, chaining clusters but if two clusters coexist in proximate and/or noisy regions of the pattern space it has a tendency to agglomerate the two clusters together without first identifying the two clusters as distinct (see Everitt (1993) for a simple example of this).

The main problem with hierarchical techniques is computational cost of the calculation of the proximity matrix consisting of the distances between all observations and its recalculation for every new cluster. The initial proximity matrix requires $N(N-1)/2$ distance calculations. This can impose prohibitive memory loads for large data sets. For example, a 10,000 observation data set would require a matrix with about 49 million elements. Alternatively one could calculate intercluster distance on the fly without storing the proximity matrix. For large data sets however, this would still place a very heavy computational load and is probably not practical for most applications of this type. For the density linkage techniques this cost is even higher as the distance calculations between individual observations are more elaborate. In addition, both the nearest neighbour and the uniform kernel methods involve a smoothing parameter (number of s nearest neighbours in the nearest neighbour method and the radius, h, of the sphere in the uniform kernel method) which may require repeated experimentation. In general, standard hierarchical techniques are not feasible for large data sets due to the high memory and/or computational overheads. Even more recent implementations of hierarchical techniques (see Murtagh, 1983, 1992) can still impose unacceptably high computational or memory storage costs for large data sets as they generally require either $O(n^2)$ CPU costs with $O(n)$ memory storage or $O(n^2)$ memory storage with $O(n)$ CPU costs.

Technique	Main Strengths	Main Weaknesses
Single Linkage	No cluster shape restrictions	High CPU or memory costs, sensitive to noise
Complete Linkage	More robust under noisy conditions than single linkage	High CPU or memory costs, biased towards small compact clusters
Average Linkage	Can extract unequally size clusters and is robust under noisy conditions	High CPU or memory costs, biased towards hyperellipsoidal cluster shapes
Centroid Linkage	Moderately robust under noisy conditions, fastest of the hierarchical methods examined here	Relatively high CPU or memory costs, smaller clusters tend to be overshadowed by larger clusters when agglomerated
Median Linkage	Moderately robust under noisy conditions, designed to counteract the tendency of larger clusters to overshadow smaller clusters found centroid linkage	High CPU or memory costs
Ward's Method	More robust under noisy conditions than most other clusters.	High CPU or memory costs, biased towards equally sized hyperellipsoidal clusters
Minimum Spanning Tree	No cluster shape restrictions	High CPU or memory costs
Density Linkage	No cluster shape restrictions	High CPU or memory costs

Table 3.2: Summary of the main strengths and weaknesses of hierarchical clustering methods.

Table 3.2 summarises the main strengths and weaknesses of the hierarchical methods examined in this chapter. In short, if one is faced with large, noisy data sets with nonglobular clusters, standard hierarchical techniques are probably not practical. Many techniques tend to impose hyperellipsoidal structures on the data. Although single linkage and MST can model long chaining clusters, they often do not perform well under noisy conditions. Cost considerations also preclude the use of hierarchical methods for large data sets. However, if the data set is first summarised via an initial pattern partitioning where the centroids of the initial clusters are used as input to the above hierarchical methods, hierarchical analysis of large data sets is practical. The next section reviews pattern partitioning methods and is followed by a discussion of hybridising pattern partitioning and hierarchical methods for analysing large data sets.

3.4 Pattern Partitioning Techniques

Pattern partitioning techniques are characterised by the process of dividing up the observations into k mutually exclusive groups. Often k will be prespecified (*e.g.* Cheng and Milligan, 1996; Kohonen, 1995, 1982; Forgy, 1965) although there are approaches to pattern partitioning where this is not necessary (*e.g.* Fritzke, 1995, 1991; Chaudhuri *et al,* 1992; Liu and Tsai, 1989; Ball and Hall, 1965). Often, the process of allocating observations to clusters is accomplished via an objective function whose maximisation or minimisation represents an optimal (though not necessarily globally optimal) solution to the clustering problem as in k-means (Forgy, 1965) and moving methods (Ismail and Kamel, 1989). Other techniques attempt to systematically search the pattern space for areas of low or high density such as moment preserving methods (*e.g.* Liu and Tsai, 1989). However the partitioning is accomplished, the overriding characteristic of partitioning algorithms is that they are nonhierarchical: a given observation can only be classified into a single mutually exclusive cluster[9]. There are no hierarchical relationships between observations or clusters.

As the first stage of the APC methodology is an initial pattern partitioning cluster analysis of the data it is worthwhile discussing some of the various partitional

[9]Fuzzy clustering allows for multiple cluster membership. See Bezdek (1981).

techniques as a good initial clustering of the data is essential for APC. Also, some aspects of APC draw their inspiration from methods used in partitional techniques that involve density searches of the pattern space.

3.4.1 Searching all possible partitions

An intuitively simple approach to pattern partitioning would be to search every possible partition in the data set to find the one that best optimises a given objective function or some other measure of cluster suitability. However, the number of possible partitions is impractically large even for small problems. The total number of partitions that would need to be searched, *S*, is:

$$S = \frac{1}{k!}\sum_{j=1}^{k}\left(-1^{j}\binom{k}{j}(k-j)^{N}\right) \qquad 3.14$$

where *N* is the number of patterns and *k* is the number of clusters. (Anderberg, 1973). From equation 3.14 there are 10^{30} different allocations of 100 objects into 2 classes (Hand, 1981). This is obviously not practical for even small data sets although a number of techniques have been developed to lighten the computational load such as branch and bound methods (*e.g.* Massart *et al*, 1983).

Given the impracticalities of searching every possible partition, another approach is to start with a fixed number of *k* clusters and then find the *k* cluster partitioning that optimises an objective function such as the total within group error:

$$E = \sum_{j=1}^{k}\sum_{i\in C_j}\left(\mathbf{c}_j - \mathbf{x}_i\right)^2 \qquad 3.15$$

$$\mathbf{c}_j = \frac{1}{n_j} \sum_{i \in C_j} \mathbf{x}_i \qquad 3.16$$

where k is the number of clusters, n_k is the number of patterns in cluster k and $\mathbf{c}_k$ is the centroid of cluster k and $\mathbf{x}_k$ is an observation belonging to cluster k. Alternatively one can search the pattern space for areas of high or low density and then use this information to identify clusters. These two approaches will now be discussed.

3.4.2 K-means

The k-means algorithm is one of the most commonly used pattern partitioning clustering techniques. Its popularity lies in its simplicity, speed and good clustering performance. A number of studies have found it to be the best performing nonhierarchical clustering algorithm in Monte Carlo simulations (Milligan and Cooper, 1987; Bayne *et al,* 1980; Cowgill, 1993). A variety of different variations of the basic k-means algorithm exist; for a more complete discussion of k-means see Anderberg (1973) or Jain and Dubes (1988). In its simplest incarnation k-means is run by specifying *a priori k* initial clusters to be searched for. The following outline is that devised by Forgy (1965) and is one of the most straight forward k-means algorithms:

1. Begin with a set of *k* initial cluster centroids. The initial centroids (*i.e.* seeds) can either be composed of randomly chosen data observations or can be found via some "educated guessing" (e.g. Cheng and Milligan, 1996; Babu and Murty, 1993; Shattuck *et al*, 1991)10.

2. Make a pass through the entire data set allocating each observation to its nearest centroid.

3. Once all observations have been allocated to their nearest centroid, recalculate each centroid as being the mean vector of all observations allocated to that cluster.

10A survey of seeding methods is given in Appendix I.

4. Repeat steps 2 and 3 until a pass can be made throughout the entire data set where no observation changes its cluster membership.

Note that this process is the equivalent to minimising eq. (3.15) and implicitly defining a cluster as being a compact set of patterns. A shortcoming of k-means is that although it can be shown that it will converge (Anderberg, 1973; Pollard, 1981) there is no guarantee that it will converge at the optimal minima. In fact, as k-means will often converge sub-optimally, a common approach is to run k-means a large number of times with different initial seed values and then take the best solution (Backer, 1995).

Due the requirement in k-means that *k* be prespecified as well the lack of any guarantee that convergence will be global, a number of cluster splitting and merging algorithms have been proposed as modifications to the basic k-means algorithm. The simplest of these is to add an additional step to Forgy's method (Wishart, 1969b). Wishart suggests initially running k-means with a large number of *k*. Once convergence has been reached, remove any clusters whose number of member observations is below some threshold. Repeat steps 2 and 3 again until convergence and, if necessary, repeat the entire process until all clusters have the minimum necessary number of observations.

ISODATA, developed by Ball and Hall (1965) extends Wishart's approach even further. ISODATA begins as Forgy's method:

1. Seed the *k* clusters using one of the methods discussed in Appendix A1.
2. Find the closest centroid to each observation.
3. Once a pass has been made through the data set, recalculate the cluster centroids as the mean vector of all observations that belong to that cluster.
4. Repeat steps 2 and 3 until convergence or until NPARTS iterations have been completed.

5. Remove any clusters that contain fewer than *THETAN* observations. The observations contained in the removed clusters are treated as outliers and discarded for the remainder of the analysis.

6. Decide whether to test for splitting or merging the remaining clusters:
   a. Test for merging the clusters if the number of clusters is greater than twice some threshold (*NWRDSD*).
   b. Test for splitting clusters if the number of clusters is one half *NWRDSD*.
   c. Otherwise alternate between splitting and merging on different iterations.

7. Recompute centroids as in step 3.

8. Repeat steps 5, 6 and 7 for a pre specified number of iterations (ITERMAX) or until convergence has been reached.

The merging of clusters is accomplished by calculating all pair wise distances between clusters. If any of the intercluster distances are less than some threshold (*THETAC*), the two centroids are merged. The new intercluster distances between the new centroid and all other centroids is then calculated and this process is repeated until either there are no new merges or until a pre specified number of merges have been completed.

Clusters are split if the within cluster standard deviation for any variable is greater than some threshold. This threshold, *THATAE*, is derived from the original standard deviation of the given variable on the entire data set. The two new clusters are created by dividing up the observations belong to the original variable along the mean value of the original variable. The two resulting centroids are computed and if the distance between the two centroids is greater than *THETAAC*, the split is maintained, otherwise the original cluster is kept.

The main difficulty with ISODATA is that it requires a lot of user interaction. There are also seven user defined parameters (summarised in Table 3.3) which could be a bit daunting for many data exploratory exercises.

Parameter	**Definition**
k	Initial number of clusters.
THETAN	Minimum number of patterns that must be in a cluster.
NWRDSD	Threshold of the number of clusters used to determine whether splitting or merging of clusters is to be tested for.
THETAC	Merge clusters if the intercluster distance is less than THETAC
THATAE	Variable standard deviation threshold used for determining whether a cluster should be split.
NPARTS	Max iterations for clusters to converge within a merging/splitting iteration.
ITERMAX	Max iterations for the splitting/merging process.

Table 3.3: Summary of parameters used in ISODATA.

Another variation of k-means has been proposed by Cheng and Milligan (1996). Under the assumption that outliers can lead to k-means converging to sub-optimal minima, Cheng and Milligan proposed an *influence detection* routine to eliminate individual observations that significantly effect convergence. Once the outliers are removed, convergence to high density areas should be enhanced. The influence detection method is quite simple. First, k-means is run until convergence. Next, for each observation, remove the observation from the data set and re-run k-means. Using the Adjusted Rand Index (see Chap 5) measure the degree of agreement between the previous cluster solution and the new one with the object removed. If the clustering differs significantly, the given observation is an outlier and can be removed. The computational cost of running k-means once for every observation may be quire high (requiring at least *2N* passes through the data set) so this approach may only be practical for small or moderately sized data sets.

3.4.3 Moving Methods

An approach to partitional clustering similar to k-means has been proposed by Duda and Hart (1973) and Ismail and Kamel (1989) called the moving method. A number of different variations of the moving method exist although Zhang and Boyle (1991) have conducted some empirical comparisons suggesting that all perform equally well.

The basic idea behind the moving method is to begin with a fixed value of *k* and then pass through the data set and attempt to move each observation to another cluster. If this move results in the further minimisation of a cost function keep the move. The process of moving observations to more appropriate clusters is continued over a number of passes through the data set until convergence has been achieved. As with k-means, convergence is defined as the point at which no observation can be moved to any other cluster and improve the cost function. The various moving method algorithms differ in how a more optimal cluster is sought for each observation. The simplest is to keep the first move that reduces the cost function. Another is to test the move at each possible cluster and then keep the move leading to the greatest reduction in the cost function. As both of these techniques tend to follow different convergence paths, Ismail and Kamel recommend alternating between the two strategies on every observation. Zhang and Boyle however, found that all three strategies work equally well.

The cost function used in the moving method is the same as that for k-means (eq. 3.15). When attempting to move observations to other clusters, the effects of the change on the cost function can be easily calculated.

Given the error due to cluster *j*:

$$E_j = \sum_{i \in C_j} (\mathbf{c}_j - \mathbf{x}_i)^2 \tag{3.17}$$

moving a pattern $\mathbf{x}_i$ from cluster *j* to cluster *m* and adjusting the cluster centroids accordingly, the change in the error due to cluster *j*, $\Delta^{-}E_j$, will be:

$$\Delta^{-}E_j = \frac{n_j(\mathbf{c}_j - \mathbf{x}_i)^2}{(n_j - 1)} \tag{3.18}$$

and the change in the error due to cluster m, $\Delta^{+}E_m$, will be:

$$\Delta^{+}E_m = \frac{n_m(\mathbf{c}_m - \mathbf{x}_i)^2}{(n_m + 1)} \qquad 3.19$$

if $\Delta^{-}E_j > \Delta^{+}E_m$ the move decreases the total cost function.

Zhang and Boyle show that the convergence states of the moving method are also the convergence states of k-means, however not all convergence states of k-means are necessarily the convergence states of the moving method. For each pattern, $\mathbf{x}_i$, k-means checks to see if

$$\begin{gathered} \left|\mathbf{x}_i - \mathbf{c}_j\right|^2 \le \left|\mathbf{x}_i - \bar{\mathbf{x}}_m\right|^2 \\ m = 1, 2, \ldots, k. \\ \mathbf{x}_i \in C_j \end{gathered} \qquad 3.20$$

for $m \neq j$, $m = 1, 2, \ldots, k$

is true. Convergence is reached when 3.20 is true for all patterns. For the moving method, convergence is improved when

$$\begin{gathered} \Delta^{-}E_j > \Delta^{+}E_m \\ \textit{if } x_i \in C_j \end{gathered} \qquad 3.21$$

and when convergence is reached, each pattern will satisfy

$$\begin{gathered} \frac{n_j \left|\mathbf{x}_i - \mathbf{c}_j\right|^2}{n_j - 1} < \frac{n_m \left|\mathbf{x}_i - \mathbf{c}_m\right|^2}{n_m + 1} \\ \textit{or} \\ \left|\mathbf{x}_i - \mathbf{c}_j\right|^2 < \frac{n_m(n_j - 1)}{n_j(n_m + 1)} \left|\mathbf{x}_i - \mathbf{c}_m\right|^2 < \left|\mathbf{x}_i - \mathbf{c}_m\right|^2 \end{gathered} \qquad 3.22$$

for $m \neq j$, $m = 1, 2, \ldots, k$

if $\mathbf{x}_i \in C_j$

Any pattern that satisfies eq. (3.22) will also satisfy eq. (3.20). However, not all patterns that satisfy eq. (3.20) will hold for eq. (3.22). Therefore, although all the convergence states of k-means are those of the moving methods, not necessarily all moving method convergence states can be accessed by k-means. From this Zhang and Boyle suggest that if a more optimal convergence state of the moving method is not a convergence state of k-means, the moving method will perform better. However, it could also be the case that the extra convergence states correspond to local minima, which could increase the likelihood of the algorithm getting stuck before it reaches a more optimal state. Ismail and Kamel (1989) and Zhang and Boyle (1991) conducted empirical simulations suggesting that the moving method does converge better than k-means. However, as these experiments were limited to only a few very low dimensional data sets, a more comprehensive and controlled evaluation would be more informative of the relative clustering abilities of k-means and the moving method. This is examined further in Chapter 6.

3.4.4 Moment Preserving Methods

An entirely different approach to partitional clustering is moment preserving clustering (Liu and Tsai, 1989). Moment preserving clustering is based on the projection method (PM) developed by Henrichen and Fu (1968). PM uses Karhunen-Loeve expansion (also known as principal component analysis) to find a set of orthogonal axes of the data set. The data is then projected onto these axes and the resulting marginal densities are examined. The local minima of the first axis is then used to partition the data into clusters. If no minima is found, the next orthogonal axis is used. This process is then repeated on each of the previously discovered partitions until all of marginal densities found by projecting on to the orthogonal axis are found to be unimodal.

A problem with this technique, as pointed out by Liu and Tsai (1989) is that clusters that overlap on the projected axes can obscure their separability. Figure 3.2 shows an example. Extracting the two axis and then looking at the marginal densities of the projected data does not reveal the fact that two clusters are present in the data. Using the Henrichon and Fu approach, one would be led to conclude that the data set was unimodal.

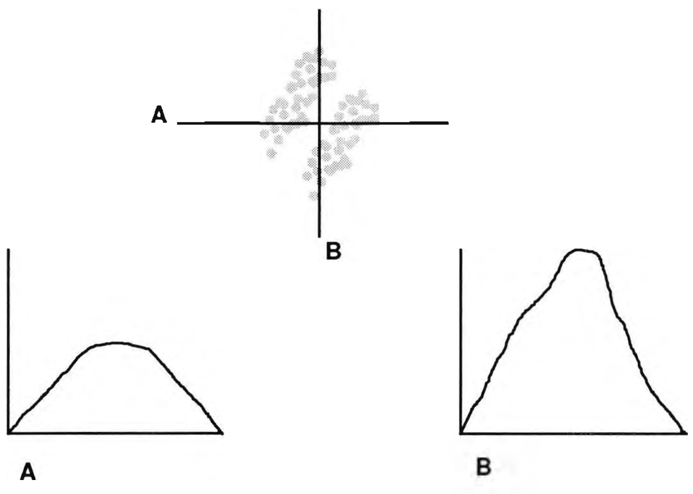


Figure 3.2: Potential problem with moment preserving clustering.

Liu and Tsai attempt to circumvent this problem by searching the intercluster regions directly for areas of low density. Their algorithm can be summarised as follows:

STAGE 1:

step 1: Generate d orthogonal eigenvectors ($\mathbf{u}_1$, $\mathbf{u}_2$, ... $\mathbf{u}_d$) from the covariance matrix of the data.

step 2: Divide the pattern space up into subregions by segmenting the axis corresponding to the largest eigenvector of the covariance matrix into k_d subsections of length b_d within the range of the data. A technique of determining , k_d and b_d is given in Liu and Tsai (1989).

step 3: Search for sparsely populated subregions whose pattern density is below some threshold, f_t. If none can be found then either:

3.1 Try reducing the range of the data used along each axis.

3.2 Return to step 2 and use the next largest eigenvector.

3.3 If all eigenvectors have been used, go to STAGE 2.

step 4: For each consecutive set of subregions with densities below f_t take the smallest (*i.e.* least dense) one or the one that is not one of the two end regions and bisect it with a hyperplane orthogonal to $\mathbf{u}_d$ to partition the data.

STAGE 2: For each partition found, repeat STAGE 1 using the data contained in that partition.

STAGE 3: For each pair of single class partitions resulting from *different* executions of step 4, take the union of each pair and repeat stage 1 to ensure there are sparse regions in the union. If not, merge the two partitions together.

The key feature of Liu and Tai's method is the reduction of the size of the subregions being searched in step 3, which, as can be seen in Figure 3.3 increases the likelihood of detecting overlapping clusters. Once a sparsely populated region is found, a hyperplane perpendicular to the eigenvector bisecting the sparse region is created to divide the patterns into separate groups. The whole search process is repeated recursively on each cluster discovered until no further clusterings are found. If in the process of separating clusters a cluster is bisected by a hyperplane, the merging step will reconstruct the cluster.

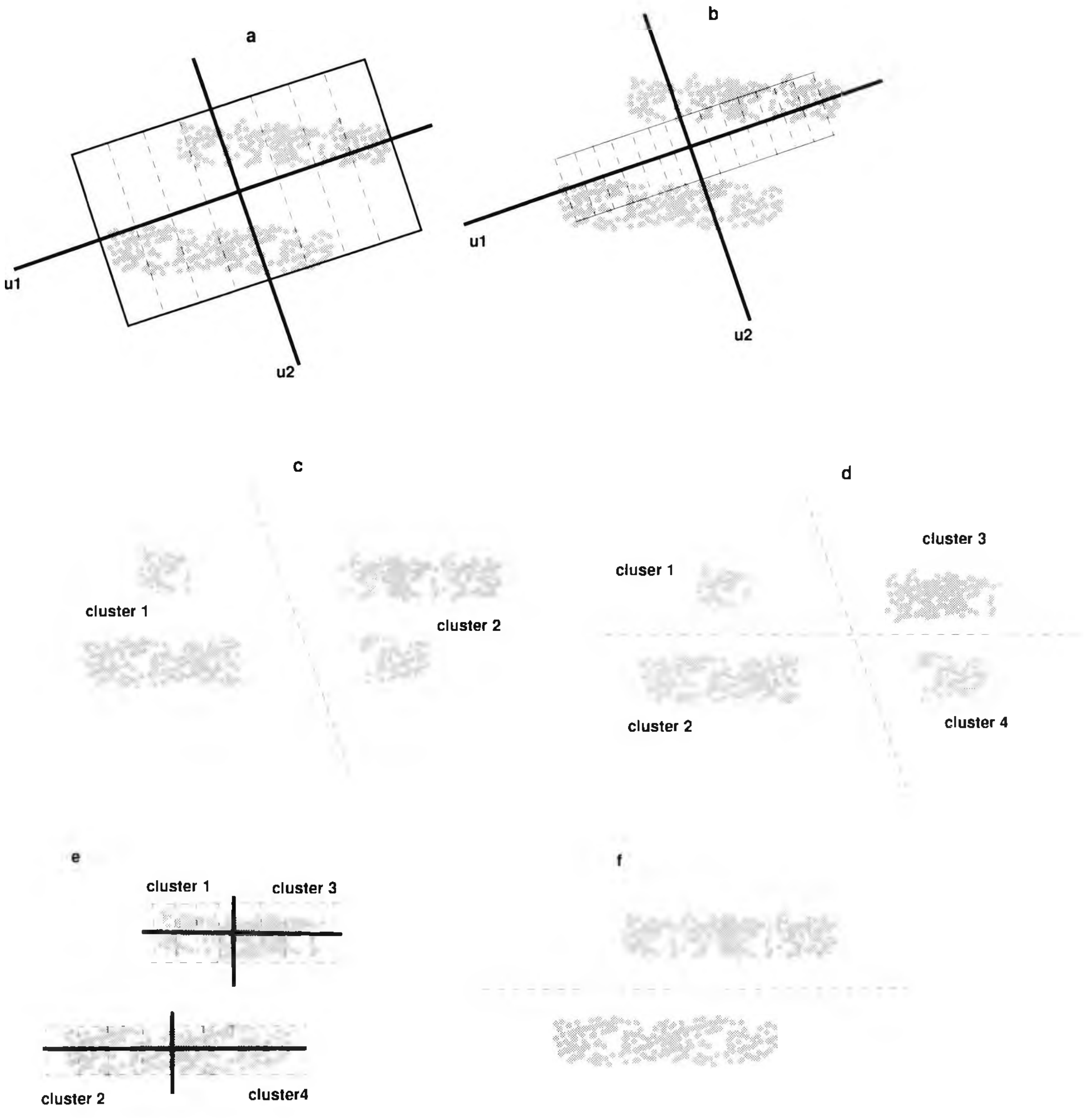


Figure 3.3: The Liu and Tsai (1989) approach begins with searching subregions along the axis determined by the largest eigenvector, u1(a). If no sparsely populated subregion is found, the size of the subregions is reduced which increases the likelihood of finding sparse areas. This is shown in (b) where reducing the size of the subregions has led to three sparsely populated subregions near the intersection of u1 and u2. The data is then partitioned along the line the bisects the subregions orthogonal to u1. This process is repeated on the two resulting partitions which generates the four clusters shown in (d). As no further division of these clusters is possible, each pair of clusters is tested for merging. This leads to clusters 1 and 3 and clusters 2 and 4 being merged as when these clusters are tested using the same process used in stage 1 no partition is found to exist (e). The final result is the data partitioned into the two original hyperellipsoidal clusters (f).

The main shortcoming of Liu and Tsai's approach is that it is unlikely to perform well unless the clusters are well separated and hyperellipsoidal in structure as it relies on a linear transformation of the data. The general concept of directly searching the intercluster regions for low density areas by dividing up the pattern space into discrete regions also forms the basis of APC. However, as will be seen in Chapter 4, APC uses a different approach than Liu and Tsai to determining which subregions are to be searched for areas of high/low density. APC also uses a different measure of density as well. Finally, APC generates a hierarchical clustering of the data, unlike the case of Liu and Tsai's algorithm.

3.4.5 Chaudhuri *et al* (1992)

Chaudhuri *et al* (1992) introduced another splitting and merging method similar in approach as Liu and Tsai where the splitting and merging of clusters is based on destiny searches of the intercluster regions. Chaudhuri *et al* propose two basic approaches to partitioning the data: type-I splitting and type-II splitting.

Type-I splitting involves the search of strips in the pattern space in different directions around previously found clusters or, in the case of the initial application of their clustering algorithm to the data, the centroid of the data. The data is partitioned along strips of low density found in the data space. Given a d-dimensional data set, strips along each of the $2^{d-1} + d$ directions from the cluster centroid are searched. For example, in a two dimensional data set, 1 horizontal, 1 vertical and 2 diagonal strips are searched which correspond to the d axis and the diagonals (Figure 3.4).

The widths of the strips need also to be determined. If too large, the intercluster densities will be over estimated, too small and the density estimate will be become unreliable due to the small sample taken[11]. In any case, if the number of points in one of strips is less than a predetermined threshold, l_l, the data is partitioned along that strip.

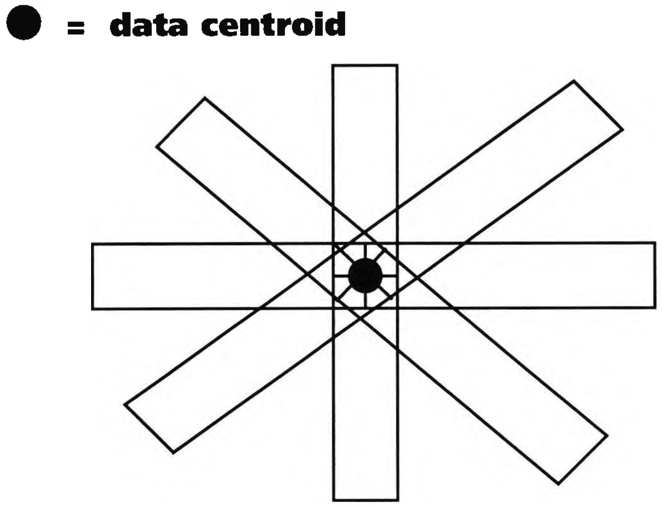


Figure 3.4: Type-I splitting involves searching strips of the pattern space for areas of low density. 4 strips are constructed corresponding the $2^{d-1} + d$ directions from the cluster centroid to be searched.

In Type - II splitting Forgy's k-means with $k = 2$ is applied to the data. Let $\mathbf{c}_1$ and $\mathbf{c}_2$ be the centroids of the newly created clusters C_1 and C_2 after the application of k-means and let $d_{1,2} = \| \mathbf{c}_1 - \mathbf{c}_2 \|$ be the distance between the two centroids. Chaudhuri *et al* next define a set of points, A, called the *almost equidistant point set* which is the set of patterns where the difference, z_n, between each pattern's distance from $\mathbf{c}_1$ and $\mathbf{c}_2$ is less than 10% of $d_{1,2}$:

$$z_n = | \, \| \mathbf{x}_n - \mathbf{c}_1 \| - \| \mathbf{x}_n - \mathbf{c}_2 \| \, |$$

if $\mathbf{x}_n \in \{C_1 \cup C_2\}$

and

[11]The reader is referred to Chaudhuri *et al* (1992) for further discussion of the strip width determination algorithm.

$A = \{ z_n < d_{1,2}/10 \}$ 3.23

If $(n_A/n) \times 100 < l_2$ 3.24

then C_1 and C_2 are treated as two separate clusters where l_2 is a predetermined threshold. If equation 3.15 is not satisfied, the two clusters, C_1 and C_2 are considered for re-merging using a process that takes into account the degree to which the boundary points between clusters are equally distributed between clusters (see Chaudhuri *et al* (1992) for further details).

The Chaudhuri *et al* algorithm can be summarised as follows:

Let K_J be the number of clusters at iteration J and initialise the algorithm with $J = 0$ and K = 1.

1. Apply type-I splitting to every cluster. If $J = 1$, use the centroid of the data.
2. If a cluster is split, set $J = J + 1$ and go to step 1.
3. Apply type-II splitting to each cluster. If any clusters are split, go to step 4, otherwise go to step 5.
4. Apply type-I splitting to each pair of clusters (one at a time) found in step 3.
    - 4a. If neither are split further, test the pair for remerging using the equidistant point set (*A*) found for those two clusters in step 3.
    - 4b. If at least of the of the two clusters are split, check the merging restriction on the pair if *A* contains patterns from both clusters.
5. If $K_J = K_{J-1}$ stop. Otherwise, set J = J + 1 and go to step 1.

The main drawback to the Chaudhuri *et al* algorithm is scalability with the dimensionality of the data. Figure 3.5 plots the number of dimensions of the data with the corresponding number of strips that need to be searched. Type-I splitting requires ($2^{d-1} + d$) strips to be searched - one for each axis and one for each diagonal. For low dimensional data sets this is feasible as d = 2 or d = 3 only require 4 and 7 strips each. However, for $d = 20$, there are 524,308 strips to be searched for each cluster. At $d = 50$, this figure is on the order of 10^{14}. Therefore, the use of this algorithm on large data sets with significant numbers of variables is costly in terms of CPU expenditure.

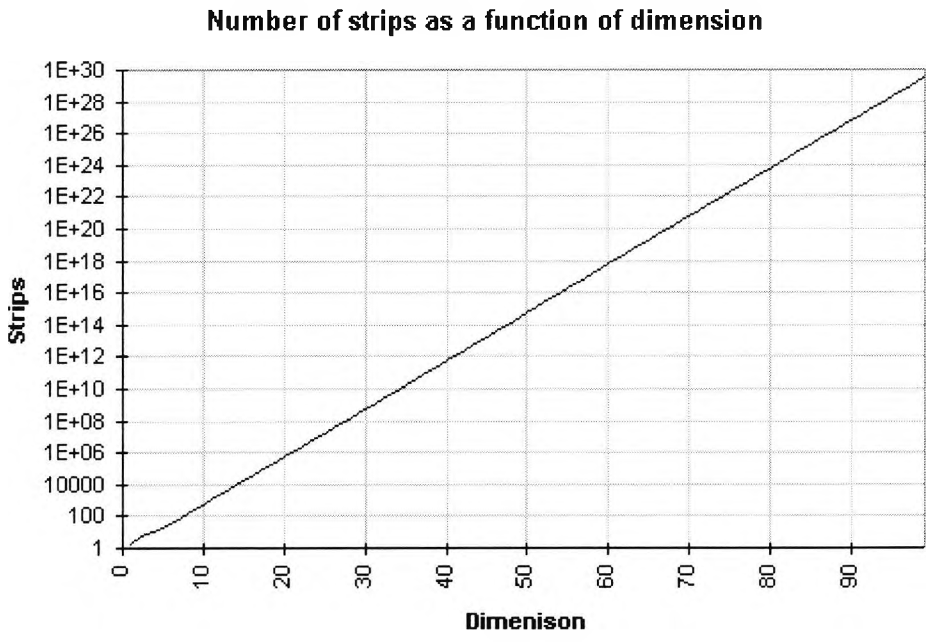


Figure 3.5: The number of search strips needed in the Chaudhuri *et al* (1992) algorithm as a function of dimensionality. Each cluster in the data requires the above number of strips to be searched. Note that the x axis is *log* scale.

3.4.6 Unsupervised Neural Networks

Self-organising neural networks are generally used for clustering, classification, data reduction and data visualisation. Although there exists a wide range of self-organising neural network architectures (*e.g.* GTM: Bishop, 1997; TS-SOM:

Koikkalainen, 1994, 1995; growing grid networks: Fritzke, 1991, 1995; Fuzzy ART: Carpenter *et al,* 1991; ART2: Carpenter and Grossberg, 1987) only the more commonly used Kohonen Self-organising neural network (KSONN) and "winner take all" self-organising neural network (WTANN) architectures are discussed here as these represent the basic algorithms from which most other variations are built. Both of these networks consist of an input layer of *d* units that take on the values of the *d*-dimensional data vector and a one, two or occasionally three dimensional output layer. Each output unit is fully connected to the input layer but there are no interoutput or interinput unit connections (Figure 3.6). For KSONNs the idea is to train the network such that similar clusterings in the data correspond to proximate units in the output layer so that the user is left with a graphical representation of the structure of the data. For this reason, the output layer of the KSONN is often referred to as a topological feature map as clusters with similar features will be represented in closer proximity to each other in the output layer than clusters with more dissimilar features. For WTANNs the proximity constraint is dropped and no relationship between clusters is implied by their proximal location in the output layer. The basic algorithm behind KSONNs and WTANNs will now be briefly outlined, for a more thorough discussion see Kohonen (1995, 1982), Haykin (1994), Hertz *et al* (1991) or Rumelhart and Zipser (1986).

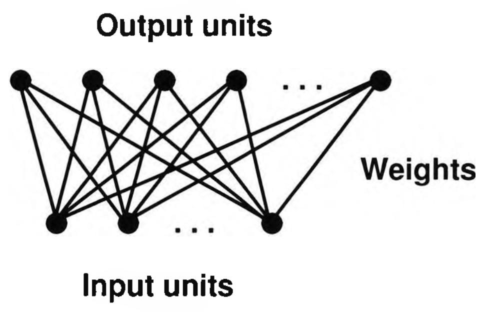


Figure 3.6: Architecture of an unsupervised neural network.

The training algorithm itself is fairly straight forward: each pattern is presented to the input layer and the distance between the pattern and the weights of each of the output units is calculated. The output unit with the closest weights to the input pattern has its weights updated such that they are fractionally closer (determined by the learning rate) to the input pattern than they were before. In order to achieve the topological feature map effect in the KSONN output layer, not only is the closest unit updated, but so are its nearest neighbours in the output unit grid. The size of the update neighbourhood usually starts out quite large (*i.e.* 2/3 or more of the output layer (Haykin, 1994), and is gradually reduced to 0 during training. The training algorithm is an iterative scheme that may require many passes through the entire training data set in order to provide a good mapping of the data. The number of passes made through the data (epochs) must also be pre set prior to training as the learning rate and neighbourhood function are generally calculated based on the current number of epochs.

More formally, let **X** be set of *N*, *d*-dimensional training data patterns ($\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots \mathbf{x}_N]$),] , and **W** be the set of weights connecting the *o* output units to *d* input units where $\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, \dots \mathbf{w}_o]$) and $\mathbf{w}_o = [w_{o1}, w_{o2}, \dots w_{od}]$.

1. Set $t = 0$, $T = e{*}N$ where *e* is the number of epochs to be run and initialise **W** to small random values.

2. Randomly select an input pattern, $\mathbf{x}_n$, and present it to the network.

3. Calculate the distance between the input pattern to the weight vector of each output unit where distance is calculated as:

$$d_0(\mathbf{x}_n) = ||\mathbf{w}_0 - \mathbf{x}_n|| \qquad 3.25$$

   where || || is the Euclidean distance.

4. Find the output unit , *o**, whose weight vector is closest to the input vector such that:

$$||\mathbf{w}_{0*} - \mathbf{x}_n|| \leq ||\mathbf{w}_0 - \mathbf{x}_n|| \text{ for all } o. \qquad 3.26$$

5. Next, update the weight vectors :

$$\mathbf{w}_o(t+1) = \mathbf{w}_o(t) + \Lambda(o,o^*)\eta[\mathbf{x}_n(t) - \mathbf{w}_o(t)] \tag{3.27}$$

6. If $t = T$ stop, otherwise increment t and go to step 2.

$\Lambda(o,o^*)$ is a neighbourhood function which determines how many neighbouring units along with the winning unit have their weights updated. Although $\Lambda(o,o^*)$ can take a variety of forms (see Nour and Madey (1996), Haykin (1994) and Balakrishnan *et al*, (1994) for discussion) for the purposes of this thesis it is defined as returning a value of 1 for o^* and all units within a radius r of o^*, and returning 0 for all other units as this is the procedure used in Chapter 6. By gradually decreasing r over time, the topological proximity of the output units representing clusters is maintained. The learning rate η, determines the degree of weight change with each pattern. Generally η is initialised in the range of $(0 < \eta \leq 1)$ and then gradually reduced so as to reach 0 at the end of training (*i.e.* when $t = T$). The larger initial value of η allows the network to map the gross structure of the data in the output layer while the smaller values towards the end of training enables the network to "fine tune" the output unit weights.

If one is not particularly interested in a topological mapping, the neighbourhood function can be dispensed with so that only the winning unit's weights are updated as in Rumelhart and Zipser (1986). Here the weight update rule is:

$$\mathbf{w}_o(t+1) = \begin{cases} \mathbf{w}_o(t) + \eta[\mathbf{x}_n(t) - \mathbf{w}_o(t)] & \text{if } o = o^* \\ \mathbf{w}_o(t) & \text{otherwise} \end{cases} \tag{3.28}$$

These networks, referred to as "winner take all" neural networks (WTANN) train significantly faster as there are fewer weight updates to be made. In addition, as one is not producing a topological feature map of the data, only a one dimensional output layer is used.

The main drawback to both KSONN and WTANN is that they require many passes through the data and are therefore computationally expensive to run compared to k-means or the moving methods. In fact, one can view KSONN and WTANN as variations of k-means and the moving method. K-means, the moving method, KSONN and WTANN all attempt to cluster the data by partitioning the observations into a fixed number of *k*, mutually exclusive clusters. They differ largely in how the partitioning is achieved. During each pass through the data set, k-means finds the closest cluster "prototype" to each observation. Once this has been found for each observation, the prototypes are recalculated as the mean vector of all observation closest to that prototype. The moving method is similar except that the prototypes (*i.e.* mean vectors) are recalculated with every pattern if that pattern is closer to a prototype vector other than the one corresponding to the cluster to which it already belongs. Both k-means and the moving method are said to have converged when a pass can be made through the data set with no observation changing its cluster membership. Both KSONN and WTANN also update the prototype vectors with each pattern except that rather than recalculating the prototypes as being the mean vectors of all observations that belong to that cluster, the prototype vector is updated so as to be fractionally closer to the observation being presented to the network as in eq. (3.27) for KSONN and eq. (3.28) for WTANN. The primary difference between KSONN and WTANN is that with KSONN both the closest prototype vector *and* its neighbours (in the output layer) as determined by $\Lambda(o,o^*)$ are updated.

Technique	Main Strengths	Main Weaknesses
K-means	Low CPU and memory costs	Biased towards hyperellipsoidal cluster shapes
Moving methods	Low CPU and memory costs	Biased towards hyperellipsoidal cluster shapes
Moment Preserving	Capable of distinguishing clusters that overlap on the principal axes of the data	Clusters must be well separated, biased towards hyperellipsoidal cluster shapes
Chaudhuri *et al* (1992)	No cluster shape restrictions	CPU costs increase exponentially with the dimensionality of the data
Unsupervised Neural Networks (KSONN and WTANN)	Low memory costs, topological map of KSONN provides an easy to use representation of relationships between clusters	High CPU costs

Table 3.4 Main strengths and weaknesses of the pattern partitioning methods discussed in this chapter.

3.4.7 Other Approaches

A summary of the main strengths and weaknesses of the pattern partitioning techniques discussed can be found in Table 3.4. Although on their own none of these methods is applicable to large, noisy data sets with non compactly shaped clusters, a number of them, particularly k-means and the moving method, could be utilised as the initial pattern partitioning component of APC. It should also be pointed out that a wide range of other partitioning methods exist - far too many to be surveyed in depth here (see Arabie and Hubert (1996) for review). Recent advances in pattern partitioning include mean-tracking (Sutanto *et al,* 1997) which uses a moving window over the data to directly search for modal areas, evolutionary computing based optimisation strategies (Babu and Murty, 1993,1994; Srikanth *et al,* 1995), simulated annealing (Klein and Dubes, 1989), tabu search (Al-Sultan, 1995) and various heuristic approaches (*e.g.* Kant *et al,* 1994; Messatfa, 1992; Gupta and

Tammana, 1995). However, due to computational costs, none of these approaches are applicable to fast clustering of large data sets within the hybrid clustering context being explored in this work.

3.5 Hybrid Techniques

In order to deal with problems related to larger data sets hybridisations of standard clustering methods have been proposed. Hybrid methods are characterised by their use of multiple clustering techniques to solve the clustering problem. Beale (1969) and Wishart (1978) first proposed the use of hybrid or multistep methods to aid in the use of hierarchical techniques with large data sets. As mentioned earlier, hierarchical techniques themselves are too inefficient to be used with large data sets. Beale and Wishart suggested the use of k-means or other pattern partitioning methods to first divide up the data set. These sub clusters can then be clustered together using hierarchical methods. Murtagh (1995) has also done similar work with self organising neural networks. This section will briefly review hybrid methods of this type - the use of pattern partitioning techniques to generate an initial partition which is then agglomerated using hierarchical techniques.

3.5.1 Two Stage Hybridisations

The two stage process proposed by Beale and Wishart is fairly straight forward. First, a k cluster partition is generated using a fast partitioning method such as k-means. Once these clusters have been identified, the centroid, $\mathbf{c}_k$, of each cluster is calculated which acts as a "feature" or "prototype" of the observations within that cluster. The k cluster centroids can now be used as inputs to hierarchical methods that would not have originally been able to deal with the large size of the raw data set. As far as the author of this work is aware, no systematic empirical comparison of the use of different standard hierarchical algorithms in a hybrid context such as this have been published.

3.5.2 Contiguity Constrained Clustering

Contiguity constrained clustering (CCC) refers to the use of clustering techniques with the constraint that cluster agglomerations can only occur between clusters that are near or "contiguous" to each other (see Hartigan (1975) or Gordon (1981) for reviews). Murtagh (1995) applied CCC to the feature maps of KSONNs. Murtagh's approach was to train a large (*e.g.* 50 x 50 unit output layer) KSONN and then generate a hierarchical clustering of the output units. As the feature map places similarly responding units together (*i.e.* similar clusters are placed in proximity to each other on the feature map) it seems natural to place a contiguity constraint on their agglomeration. Murtagh therefore constrained the agglomerations to neighbouring units (a given unit and its 8 neighbouring units assuming a two dimensional feature map or output layer). Intercluster distance between two output units, *i* and *j* is measured based on the corresponding centroids $\mathbf{c}_i$, and $\mathbf{c}_j$ of the patterns belonging to these units. Murtagh examined two distance measures one equivalent to centroid linkage:

$$d^*(C_i,C_j) = \left(\bar{\mathbf{x}}_i - \bar{\mathbf{x}}_j\right)^2 \qquad 3.29$$

and another minimum variance measure:

$$d(C_i,C_j) = \frac{n_i n_j}{n_i + n_j}\left(\bar{\mathbf{x}}_i - \bar{\mathbf{x}}_j\right)^2 \qquad 3.30$$

Once two clusters are agglomerated, the centroid of the new cluster is calculated and treated as a single object. A potential problem with this approach however, is that on many problems KSONNs require a large number of passes through the data set and may therefore be too slow for large databases.

3.5.3 Wong's Hybrid Method

Wong (1982) proposed another variation of the two step hybrid hierarchical / partitional method for data clustering. The first step of Wong's Hybrid Method (WHM) consists of applying k-means to the data to generate an initial partition as with Beale (1969) and Wishart (1978). The second step consists of the agglomeration of the initial clusters with single linkage using an estimate of the density at the midpoint between clusters as a distance metric instead of the conventional Euclidean or correlational distance metrics used in standard hierarchical cluster analysis. The distance between two clusters, C_i and C_j, is defined as:

$$d^*(C_i, C_j) = \frac{(W_i + W_j + (n_i + n_j)\|\bar{\mathbf{x}}_i - \bar{\mathbf{x}}_j\|^2/4)^{d/2}}{(n_i + n_j)^{1+d/2}} \qquad 3.31$$

where W is the within cluster sum of squares of cluster k , n_k is the number of observations is cluster k , $\|\mathbf{c}_i$ - $\mathbf{c}_j\|$ is the distance (*e.g.* Euclidean) between the two cluster centroids and d is the dimensionality of the data. Effectively, the distance between two clusters, $d^*(C_i,C_j)$, is inversely proportional to the density of observations at the midpoint between two neighbouring clusters. Two clusters, C_i and C_j, are defined as being neighbours if the midpoint $\mathbf{z}_{ij}$, between the two cluster centroids is closer to either the centroid of C_i or C_j, than to any other cluster centroid (*i.e.* if they share a boundary in the k - cluster partitioning.) The distance between two non neighbouring clusters is defined as infinite. Therefore,

$$d^*(C_i, C_j) = 1/f(\mathbf{z}_{ij}) \text{ if } C_i \text{ and } C_j \text{, are neighbours} \qquad 3.32$$
$$= \infty \text{ otherwise}$$

where $f(\mathbf{z}_{ij})$ is the density at the midpoint, $\mathbf{z}_{ij}$, between cluster centroids. The single linkage step simply treats each of the k-means derived clusters as the objects in the population and agglomerates the clusters as one would normally with single linkage with the distance between objects inside clusters defined by eq. (3.32).

WHM is both sufficiently efficient for large data sets and capable of detecting nonhyperellipsoidal cluster structures. However, Wong's intercluster distance metric based on the density at the midpoint is too insensitive to local densities of patterns between clusters. Empirical simulations in Chapter 7 indicate that WHM does not perform well under noisy conditions or when clusters lie in close proximity to each other.

APC is similar in overall structure to WHM in that it begins with an initial partition of the data that is agglomerated via single linkage using intercluster density as a distance metric. There are two significant difference between the two methods, however. First, APC uses a more direct estimate of the intercluster densities. Whereas WHM defines distance as being inversely proportional to the density at the midpoint between two clusters, APC defines distance as a function of the densities of patterns found at a number of discrete intervals between the two clusters. As will be seen, this enables APC to perform more robustly under noisy conditions. The reason for this is that in WHM, the initial k-means derived clusters are treated as a rough histogram estimation of the underlying density at each of the k clusters. This then forms the basis of density estimation at the midpoint between clusters which in turn is used as the distance metric to agglomerate the k-means derived clusters via single linkage. APC on the other hand makes a direct estimate of the intercluster density incorporating more local information in the density estimate. Although this requires more computational effort than WHM (see next chapter), the increase in the work required is compensated by the improved performance. Second, when clusters are agglomerated, APC allows agglomerations to continue to occur within previously agglomerated clusters thus producing a connected graph of the initial clusters. As is discussed below, this allows for more flexible recovery and representation of cluster shapes.

3.5.4 Two Stage Density Linkage

Two stage density linkage (Sarle, 1989) was developed as a strategy to overcome excessive chaining in single linkage. The basic strategy here is to generate modal clusters using single linkage with the restriction that two clusters are only agglomerated if at least one of them possesses *m* observations. The initial stage is stopped once all observations belong to a modal cluster. Next, the modal clusters are hierarchically agglomerated via single linkage as normal using eq. (3.4). Although less prone to false agglomerations as single linkage and more robust under noisy conditions, two stage density linkage is still at least as computationally expensive as single linkage. Moreover, empirical studies have shown that relative to other standard hierarchical clustering methods such as Ward's method the cluster recovery abilities of two stage density linkage is severely compromised by noise and moderately dispersed cluster structures (see Chen *et al,* 1995; Mangiameli *et al,* 1996)

3.5.5 An MST Based Approach to Hybrid Clustering

Chaudhuri and Chaudhuri (1995, 1997) have proposed an MST based approach to two stage hybrid clustering. As with APC, the underlying strategy is to recover nonhyperellipsoidal clusters via the agglomeration of an initial partitioning of the data. In short, their approach uses a complex initial seed selection algorithm for initialising the pattern partitioning stage where the local density of each observation (via a nearest neighbour or kernel based method) is used to determine the number and location of each seed. Once the seeds have been found the standard k-means algorithm is applied. Next, the initial clusters are agglomerated via a MST joining the clusters at the centroids. The distance between clusters (*i.e.* the edge weights in the MST) are calculated as follows: let $\mathbf{z}_{ij}$ be the midpoint between the centroids of clusters C_i and C_j, M be the set of m nearest neighbours $M = \left\{\mathbf{x}_1^{z_{ij}}, \mathbf{x}_2^{z_{ij}}, \ldots, \mathbf{x}_m^{z_{ij}}\right\}$ of $\mathbf{z}_{ij}$ and $\boldsymbol{P} = \{\mathbf{p}_{1p}, \mathbf{p}_{2p}, \ldots, \mathbf{p}_{mp}\ ;\ \mathbf{p}_{m1}, \mathbf{p}_{m2}, \ldots, \mathbf{p}_{mp}\}$, be the set of p nearest neighbours in $C_i \cup C_j$ to each observation in M. Letting $N_{ij}^k(C_l)$ be the number observations in $\mathbf{P}$ where $\mathbf{x}_m^{z_{ij}} \in C_k$ and $\mathbf{p}_{mp} \in C_l$ the distance between clusters C_i and C_j is calculated as:

$$d^*(C_i, C_j) = \left|\frac{N_{ij}^i(C_i)}{N_{ij}^i(C_j)} - \frac{N_{ij}^j(C_j)}{N_{ij}^j(C_i)} - 2\right|. \qquad 3.33$$

When the clusters should be merged

$$\frac{N_{ij}^i(C_i)}{N_{ij}^i(C_j)} \qquad 3.34$$

will be close to 1 as will

$$\frac{N_{ij}^j(C_j)}{N_{ij}^j(C_i)} \qquad 3.35$$

Therefore, the smaller the value of 3.33, the smaller the distance between clusters.

Once the MST is created using 3.33 to determine the edge weights, inconsistent edges can be removed to generate the final clustering of the data. Effectively, eq. (3.33) measures the degree of cluster overlap by measuring the extent to which patterns surrounding $\mathbf{z}_{ij}$ are representative of both C_i and C_j. The greater the separability, the more likely both $\mathbf{x}_m^{z_{ij}} \in C_k$ and $\mathbf{p}_{mp} \in C_k$ are true. Clusters with high separability will lead to values of 1 for equations 3.34 and 3.35. The greater the degree of cluster overlap, the smaller the value of 3.33. If the two clusters overlap each other significantly, then the corresponding edge weight will be small and the two should be considered for merging. The main problem with this approach is its computational cost. Initial seed selection requires a nearest neighbour based density estimate of the region of the pattern space around each observation. On top of this one still needs to calculate the nearest neighbours of each observation in M within the subset $\{C_i \cup C_{j'}\}$ for each possible agglomeration of clusters. For very large data sets, therefore, this approach may be a bit expensive although significantly more efficient than standard hierarchical methods.

The main strengths and weaknesses of the hybrid methods discussed in this chapter are summarised in Table 3.5. As can be seen Murtagh's approach, the Chaudhuri et al MST based approach and two stage density linkage still involve heavy computational costs. However, WHM the two stage hybridizations using standard hierarchical methods are sufficiently efficient for use on large data sets. These models will be examined empirically in Chapter 7.

Technique	**Main Strengths**	**Main Weaknesses**
Two Stage Hybridizations	Low CPU and Memory Costs	Clusters must be well separated (see chapter 7)
Murtagh's KSONN Contiguity Constrained Clustering	Low memory costs.	High CPU costs
Wong's Hybrid Method	No cluster shape restrictions, low memory and CPU costs.	Clusters must be well separated (see chapter 7)
Two Stage Density Linkage	No shape restrictions.	High CPU or memory costs, sensitive to noise, poor recovery of disperse clusters
MST Based Hybrid Clustering	No cluster shape restrictions.	High CPU costs

Table 3.5: Main strengths and weaknesses of the hybrid methods.

3.6 Summary

The primary purpose of this chapter was to discuss and evaluate some commonly used clustering methods assuming one is faced with applications involving large and noisy data sets where clusters can take on any arbitrary shape. It is argued that standard hierarchical techniques are generally too inefficient for use under these conditions. Heavy memory requirements and the calculation of all interobservation distances are impractical for large data bases. Moreover, most hierarchical methods tend to find globular or hyperellipsoidally shaped clusters which could be problematic if long chaining clusters are present. Single linkage and MST approaches which can extract chaining clusters tend to perform poorly under noisy conditions. Pattern partitioning techniques such as k-means and moving methods are sufficiently fast although, as they utilise the squared error clustering criterion, they are biased towards hyperellipsoidal clusters. Nonetheless, they are ideal approaches for the initial pattern partitioning stage of APC due to their computational efficiency.

Hybridisations of pattern partitioning methods and hierarchical techniques enable one to overcome some of these problems. Initially clustering the data with a fast partitioning method such as k-means and then agglomerating the resulting clusters allows one to apply hierarchical methods to large data sets. Chapter 7 empirically compares the clustering abilities of a number of hybrid methods to APC.

This chapter also discussed some other techniques that employ approaches to the clustering problem that are incorporated into APC. APC searches strips (hypercylinders, actually) of the pattern space to measure the density of observations in areas between clusters as do Chaudhuri *et al* (1992) and Liu and Tsai (1989). APC is also a hybrid method along the lines of Wong (1982), and Chaudhuri (1995, 1997).

As will be seen, in combining these approaches APC is able to both handle large, noisy data sets and detect clusters of arbitrary shape.

The next chapter will now discuss the APC methodology in detail. This will include a general overview of the proposed approach as well as in-depth discussion of the pattern partitioning stage, the measurement of intercluster density and the agglomeration stage.

4. The APC Methodology

APC is essentially a four stage process consisting of (1) an initial partitioning of the data into k mutually exclusive clusters, (2) the creation of a k x k distance matrix whose elements are calculated via a density based measure of the distance between the centroids of the initial clusters (3) the agglomeration of the initial clusters and (4) the representation of the new clusters with linked line segments. The purpose of this chapter is to give a detailed discussion of the first three of these stages in the APC methodology. The first section will discuss the general structure of APC and provide a framework with which the pattern partitioning, intercluster density estimation and agglomeration stages can be further examined. Section 2 pays particular attention to the pattern partitioning stage and discusses various algorithms that could be used for generating an initial clustering of the data. Section 3 discusses different approaches for calculating intercluster distance. Intercluster distance is measured as a function of the density of patterns found between cluster centroids. Two measures of distance will be proposed. One measures intercluster distance based on the absolute level of density between clusters which follows from the definition of a cluster as an area of high density as defined by Hartigan (1975). A second measure based on the degree to which the observations are uniformly distributed between clusters will also be introduced. Section 4 describes the agglomeration process using some simple example problems followed by an evaluation of the computational costs of APC. Although only briefly discussed here, a more detailed examination of the use of linked line segments to represent cluster structure is given in Chapter 8.

4.1 Overview of APC

Let $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\}$ be a set of d-dimensional data patterns where $\mathbf{x}_n = \{x_1\ x_2, ..., x_d\}$. APC begins with an initial clustering of **X**. The goal at this point is to locate high density areas by partitioning the data into k mutually exclusive clusters. Pattern partitioning techniques such as k-means and the moving method are ideally suited for this stage as they are computationally fast, present low demands on memory and are generally good at finding high density regions in the data (Wong, 1982). Once the k clusters have been found, the centroid, $\mathbf{c}_k$, of each cluster is calculated as:

$$\mathbf{c}_k = \frac{1}{n_k} \sum_{j=0, j \in c_k}^{n_k} \mathbf{x}_j \qquad 4.1$$

APC then proceeds to estimate the density of the areas between each of the cluster centroids to acquire a density based measure of the distance between the clusters found in the pattern partitioning stage. Using the resulting distance matrix ***D***, the initial clusters are agglomerated via the generation of a *connected graph* of the centroids using a density based measure of the distance between centroids (edge weights). Note that the agglomeration always occurs between initial cluster *centroids* and not between previously agglomerated *clusters* as in single linkage or between subtrees in MST approaches. In addition, APC uses the same neighbourhood constraint used in WHM: the distance between nonneighbouring cluster is defined as infinite. The analyst can then construct a modified dendrogram (see below) based on the connected graph to decide on a cut-off point. Any groupings of the initial pattern partitioning clusters that have been agglomerated together below this cut-off point are now treated as single clusters as in the standard hierarchical clustering model.

After the agglomeration process the newly merged clusters are modelled as the line segments connecting the centroids of the initial clusters composing the newly merged cluster. The distance of any observation from the cluster is defined as the distance (*e.g.* Euclidean) between a given observation and the closest line segment. If a particular cluster after the agglomeration process does not consist of an agglomeration of initial clusters, distance of an individual observation to the cluster is measured based on the cluster's centroid. The purpose of using linked line segments in measuring cluster membership is to take into account the shape of the cluster when making such measurements. Using the linked line segment building blocks allows the cluster shape to be modelled in a piecewise linear manner.

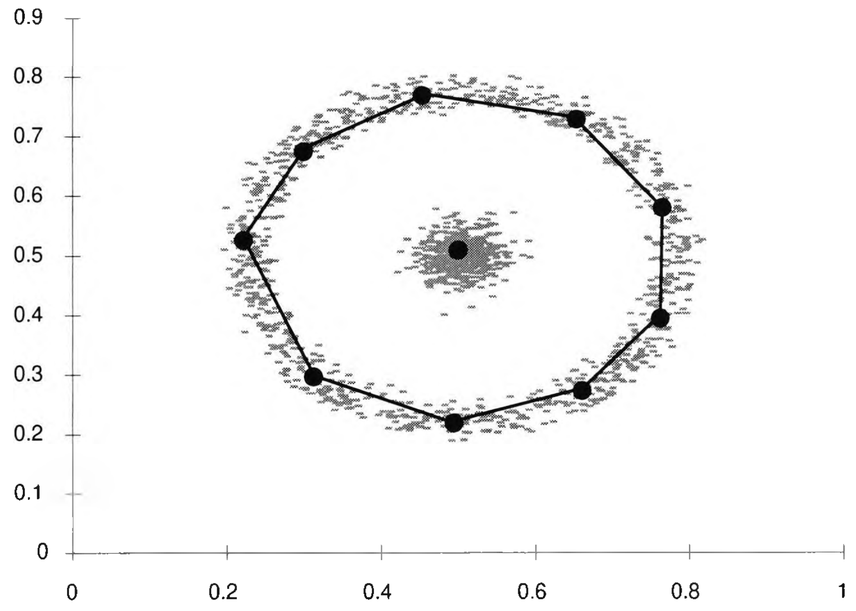


Figure 4.1: The APC solution to a two cluster problem. The heavy dots represent the initial cluster centroids found via k-means while the lines indicate agglomerations. The circular shape is represented via a piecewise linear approximation using the linked line segments.

Figure 4.1 is an example of an APC solution to a two cluster problem where one normally distributed cluster is completely enclosed by a circular chaining cluster. The heavy dots are the centroids of the clusters found in the pattern partitioning stage. The heavy lines are the links between the centroids of the initial clusters that have been agglomerated together (*i.e.* these are the linked line segment representations of the clusters). The distance of a given pattern from the central cluster is simply the distance between the pattern and the cluster centroid. For the circular cluster, distance is measured as the distance between the pattern and the nearest line segment. Note that by using a connected graph to represent an agglomerated cluster, the full circular shape of cluster in Figure 4.1 can be modelled. A spanning tree or standard hierarchical representation would have left a gap in the circle as these methods do not allow agglomerations to occur between objects that have already been agglomerated together into a cluster. For example, let A, B and C be initial clusters. If A has been agglomerated to B, and B has been agglomerated (linked) to C, MST and standard hierarchical methods will not allow A to be agglomerated (linked directly) to C. Allowing agglomerations to occur between initial clusters existing *within* the same agglomerated cluster (*e.g.* allowing A to be linked directly to C) enables better representation of nonhyperspherical clusters as in Figure 4.1.

The APC methodology can be summarised as follows:

1. Divide the data set into a set of k mutually exclusive partitions $\boldsymbol{C} = \{C_1, C_2, ..., C_k\}$. This can be done via any number of clustering algorithms such as k-means, moving methods, unsupervised neural networks *etc.*

2. Calculate the intercluster distance between each neighbouring pair of the k initial clusters. The distance between any two clusters is proportional to the density of patterns between the cluster centroids:

   $$d^*(C_a, C_b) \propto f(\bar{\mathbf{x}}_a, \bar{\mathbf{x}}_b) \text{ if } C_a, C_b \text{ are neighbours}$$
   $$= \infty \quad \text{otherwise.}$$

where $f(\overline{\mathbf{x}}_a, \overline{\mathbf{x}}_b)$ is a measure of the density between cluster centroids (see below).

3. Using the resulting distance matrix, generate a connected graph of the initial cluster centroids. The connected graph can then be converted into a hierarchical tree of all the agglomerated clusters. This dendrogram is similar to what one would obtain in standard hierarchical clustering methods except that multiple connections between centroids in the same previously agglomerated cluster are allowed (see below).

4. Decide on a cut-off point in the hierarchy and represent any agglomerated clusters as being the line segments connecting the centroids of the initial clusters composing the agglomerated cluster.

The remainder of the chapter will further examine each of the stages in APC. This will include a brief discussion of alternative techniques for the pattern partitioning stage and two measures of intercluster distance. Finally, the agglomeration process will be examined more thoroughly with some simple examples provided.

4.2 The Pattern Partitioning Stage

The use of an initial pattern partitioning stage in APC is the same as that used in WHM and other hybrid methods that use k-means to generate an initial clustering of the data from which a further hierarchical clustering can be constructed. Essentially, any clustering algorithm that partitions the data into a number of high density clusters would suffice. Assuming large samples however, this initial algorithm should be computationally efficient as with k-means or moving methods. However, for small to moderate problems other techniques such as self organising neural networks could also be used.

APC is also similar to WHM is that both use density based metrics of intercluster distance. However, the difference between APC and WHM with regards to the initial partitioning lies in role the initial clusters play in determining the intercluster distance in the agglomeration stage. WHM treats the initial partitions as a density estimate of the underlying data and then uses this information to calculate the density at the midpoint between cluster centroids. This estimate of the density at the midpoint serves as the intercluster distance metric. APC on the other hand, treats the centroids simply as indicators of regions of relatively high density. Measurements of the density of patterns between clusters is made by directly examining the patterns that lie between clusters rather than inferring the density at the midpoint based on the estimated density of the initial clusters themselves as in WHM.

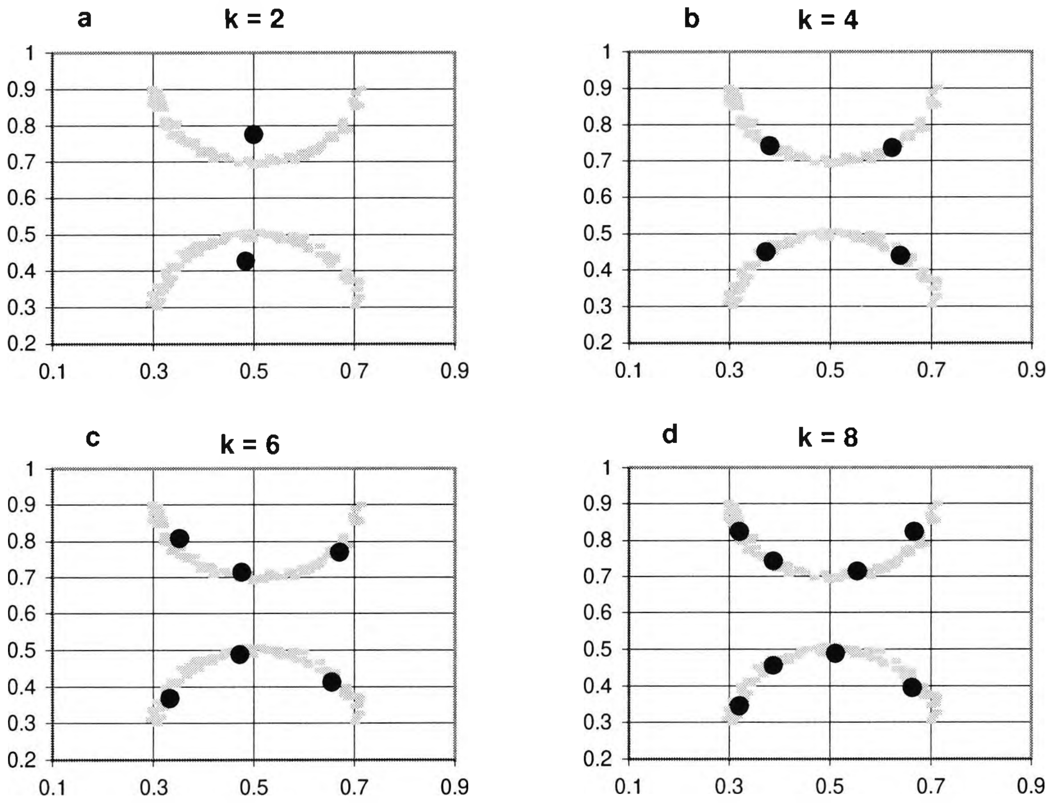


Figure 4.2: Increasing k increases the likelihood of correctly spanning the cluster shape.

Note that the assumption is made in APC that the centroids of the clusters found in the initial partitioning stage actually correspond to modal points in the data. For example, the "U" shaped clusters such as Figure 4.2*a* could lead to problems as the centroids of each cluster (k = 2) are actually in the low density areas. To overcome potential problems such as this it is recommended that larger values of k be used in APC. Figure 4.2*a* - *d* shows the location of the centroids found via k-means for $k = 2$, 4 ,6 and 8. Increasing k, increases the probability that the centroids will span the data structures. The value of k is the equivalent of a smoothing parameter; small values of k produce a rougher, more *smooth*, density estimate of the pattern space (Wong, 1982). Larger values of k give more resolution.

4.3 Measuring Intercluster Distance

The APC measure of intercluster distance is proportional to the density of the observations in the regions between neighbouring cluster centroids. The intercluster region is defined as the hypercylinder of radius s_W extending between the two centroids. Let $\mathbf{c}_a$ and $\mathbf{c}_b$ be d-dimensional vectors corresponding to the centroids of clusters C_a and C_b respectively. The intercluster density between the two cluster centroids $\mathbf{c}_a$ and $\mathbf{c}_b$ is a measure of the distribution of observations falling within the hypercylinder of radius s_W extending between $\mathbf{c}_a$ and $\mathbf{c}_b$. APC uses two different density based approaches to measure intercluster distance. The first method, which shall be referred to as the *absolute density distance* (ADD) measures intercluster distance based on the absolute level of pattern density per unit volume. The second method measures intercluster distance based on the degree of *uniformity* in the pattern density between centroids and shall be referred to as the *uniform density distance* (UDD).

4.3.1 ADD

ADD estimates the density of patterns at a discrete number, m, of connected intervals, $G = \{g_1, g_2, \ldots g_m\}$, between two cluster centroids. The interval with the lowest density value is then taken as the basis of the distance measure. Intercluster distance between clusters C_a and C_b, $d(C_a,C_b)$, is therefore defined as being inversely proportional to $min_{a,b}[f(g_i)]$:

$$ADD(C_a,C_b) = 1/min_{a,b}[f(g_i)] \quad i = 1, 2, ..., m \quad if\ min_{a,b}[f(g_i)] > 0 \qquad 4.2$$
$$\propto otherwise$$

where $f(g_i)$ is the density of the patterns in the interval g_i. This is simply a histogram based density estimate, as $f(g_i)$ is the number of patterns that fall within the interval, g_i. The higher $min_{a,b}[f(g_i)]$, the lower the distance is between the two clusters. This measure is similar to that as used in WHM except that WHM only takes into account the estimated density at the midpoint between clusters. ADD takes into account more local information that can be used to prevent the false agglomerations of clusters that are separated by narrow, low density areas. The resulting dendrogram represents a density contour tree (Hartigan, 1975) reflecting an estimate of the underlying distribution of the data. Recall Hartigan's definition of a cluster as a set of connected observations, $\mathbf{x}_{1k}$, $\mathbf{x}_{2k}$ .... $\mathbf{x}_{nk}$ where the density of a unit volume centred on each observation, $f(\mathbf{x}_{nk})$ is greater than some threshold, f_t. ADD estimates the density of a set of connected *sub regions*, g_1, g_2 ... g_m, extending between two centroids $\mathbf{c}_a$ and $\mathbf{c}_b$. $min_{a,b}[f(g_i)]$ gives the lowest density found between $\mathbf{c}_a$ and $\mathbf{c}_b$. It can be easily seen that a density contour tree of the data can be constructed from a distance matrix where distance is measured as being inversely proportional to $min[f(g_i)]$.

4.3.2 UDD

The second measure of intercluster distance, UDD, is not intended reflect the absolute density of the intercluster region, but rather the degree of *uniform* density between cluster centroids. UDD is defined as being inversely proportional to $min_{a,b}[f(g_i)]/max_{a,b}[f(g_i)]$:

$$UDD(C_a, C_b) = 1 - min_{a,b}[f(g_i)]/max_{a,b}[f(g_i)]$$
$$i = 1, 2, ..., m \quad if\ max_{a,b}[f(g_i)] > 0 \qquad 4.3$$
$$= \propto \quad otherwise$$

If each of the m intervals has the same density the ratio $min_{a,b}[f(g_i)]/max_{a,b}[f(g_i)]$ will be equal to 1. As the difference in density between the m intervals increases, the ratio will approach 0. As the higher the value of the ratio, the smaller the distance between clusters, the agglomerations between clusters are based on how uniform the point densities are between clusters. The resulting dendrogram reflects the degree to which objects reside in areas of contiguous and uniform density and is not a density contour tree such as that described by Hartigan. For example, if two clusters exist in region of low but uniform density, they will be agglomerated before two clusters in a higher but less uniform region. Figure 4.3 gives a graphic example of this. Figure 4.3*a* consists of two centroids separated by a low but fairly uniform density area. Figure 4.3*b* has a lower point density between the two centroids but is less uniform. As UDD reflects the degree of low density convexity or "gulf" separating the clusters, Figure 4.3*b* will be agglomerated before Figure 4.3*a*.

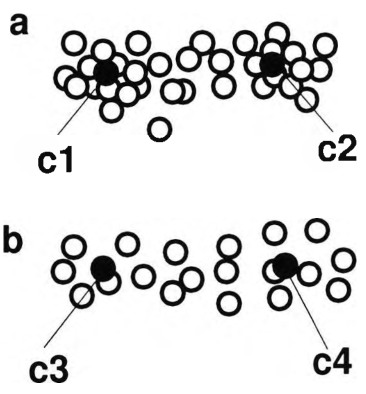


Figure 4.3: ADD and UDD will agglomerate the clusters in this figure in different orders as they measure intercluster density differently.

The advantage of this approach is that well separated clusters with relatively low density may be of importance. Since ADD emphasises the absolute level of density, it may fail to agglomerate these clusters altogether as they will only be found towards the top to the dendrogram.

As UDD incorporates a rough measure of intercluster convexity into the agglomeration process, this measure of intercluster density can also help address the question of whether a multimodal cluster is a single cluster or multiple clusters, each corresponding to one of the modes. One way of making this decision is to examine the degree of convexity between modal points. The lower the convexity, the more likely one will want to treat the multimodal structure as a single cluster. As suggested above, there may be situations where one has uniform, low density but well isolated clusters that are of interest. An intercluster distance measure based solely on the degree of density between initial clusters will fail to agglomerate these clusters until relatively late into the agglomeration process. UDD can therefore be used either in conjunction with ADD to help determine cut-off points in the dendrogram, or simply on its own. In fact, as will be seen in Chapter 7, UDD seems to produce slightly better cluster recovery than ADD.

4.3.3 Estimating Intercluster Density

At this point it is perhaps instructive to give an example of how the density estimation stage of APC is accomplished. The simplest of the density estimation techniques is the use of histograms. Regardless of whether UDD or ADD is being used, the density of the intercluster regions is calculated by examining the density of observations found in the hypercylindrical region of radius s_W extending between the cluster centroids. Again assuming a histogram based density estimate, the hypercylinder is divided up into a number of equally sized subcylinders of length s_L.

The number of observations falling within each of the subcylinders constitute the "bins" of the histogram (Figure 4.4 and Figure 4.5):

1. Provisionally connect the two cluster centroids with a line segment constituting the axis of the hypercylinder.
2. Divide the line segment into a number of sub segments (g_1, g_2, ... g_m) of length s_L.
3. Find all observations that fall within a distance of s_W of the line segment connecting the two centroids.
4. Calculate the density (number of patterns) in each of hypercylindrical subregions, defined by s_L and s_W between the centroids.
5. Find *min[f(g_i)]* and/or *max[f(g_i)]* to obtain the measure of intercluster distance.

s_W should be a function of the sizes of the two clusters being examined. For example, s_W could be calculated as being proportional to the standard deviation of the two clusters being agglomerated:

$$s_W = \alpha\left[\frac{\sigma_y + \sigma_z}{2}\right] \tag{4.4}$$

where

$$\sigma_z = \left(\frac{1}{n_z}\sum_{x_i \in z}(\mathbf{x}_i - \mathbf{c}_z)^2\right)^{1/2} \tag{4.5}$$

and α is some constant. The value of s_L should be proportional to the number of patterns in the hypercylindrical region between centroids. Evans (1983) suggests a value on the order of

$$s_L = \frac{\|\mathbf{c}_i - \mathbf{c}_j\|}{\sqrt{n_{ab}}} \tag{4.6}$$

where n_{ab} is the number of patterns in the hypercylinder between cluster centroids $\mathbf{c}_a$ and $\mathbf{c}_b$.

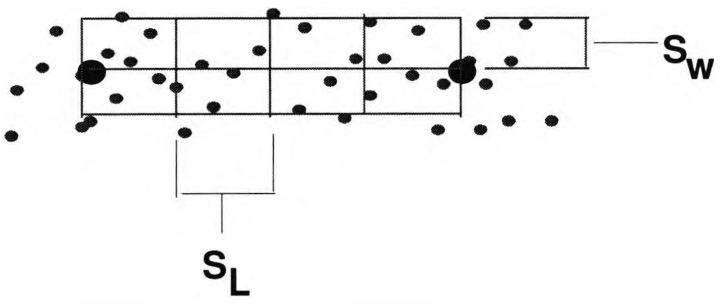


Figure 4.4: Intercluster density is estimated via a series blocks of $s_W \times s_L$ regions of the intercluster area. The heavy dots are the k-means derived centroids

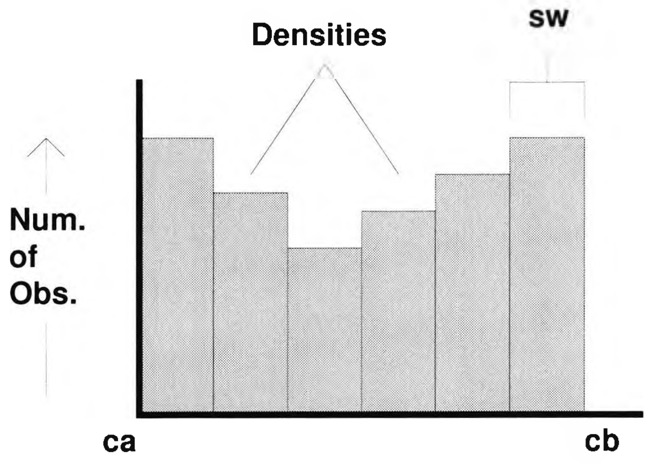


Figure 4.5: The number of patterns in each of the $s_W \times s_L$ blocks is totalled giving rise to a histogram based density estimate of the patterns between clusters.

The heights of the bars of the histogram (Figure 4.5) indicate the density in each of the subregions. Usually there will be some degree of variation in the number of observations falling within each subhypercylinder represented by the bins in the histogram. If one is interested in creating a density contour tree, ADD will give the density at the minima value (*i.e.* the smallest bin). It is this minima value that is used to measure intercluster distance. The greater the minima, the smaller the intercluster distance. Alternatively, one can look at both the minimum and maximum bins and obtain a measure of the distributional convexity of the patterns between the two cluster centroids. The smaller the difference between the maximum and minimum bins, the greater the degree of distributional uniformity.

It is not absolutely necessary that histograms be used in the estimation of the density of patterns between centroids. Other methods such as kernel estimators could also be implemented. Appendix A2 discusses and evaluates the use of a number of other nonparametric density estimation approaches that could be used in generating a density based measure of intercluster distance. The remainder of this section will discuss the use of the histogram approach in more detail.

4.3.4 Calculating a Histogram Based Intercluster Density Estimate

The version of APC evaluated in this thesis uses the simplest of density estimation techniques, histograms, to estimate the intercluster density. As discussed in Appendix A2 there are other approaches to density estimation that could be used at this stage. However, as the histogram method of estimating the intercluster density is easy to implement and computationally inexpensive, it is the only one used in this thesis. Appendix A2 discusses other density estimation methods that can be used in this stage of APC.

Given a univariate data set, a histogram density estimate of the underlying patterns can be found by dividing the data set into m equally sized intervals $G = \{g_1, g_2, \dots g_i\}$, and counting the number of data points falling within each interval as in Figure 4.6. More formally, let $h = |\max[\mathbf{x}_N] - \min[\mathbf{x}_N]|/m$ be the size of the intervals, referred to as *bins*. The density at any given point $(\mathbf{x}_N)$ is defined as:

$$f(x_N) = \frac{n_{g_x}}{N} \qquad 4.7$$

where n_{gx} is the number of observations in the same bin as $\mathbf{x}_N$. The choice of the size of the bin width can have profound effects on the resulting density estimate as can be seen in Figure 4.6. Figure 4.6*b* used a very small value for *h* while Figure 4.6*c* used a larger one. Increasing the size of the size of the bin width smoothes the density estimate. Decreasing the size of *h* makes the histogram more sensitive to local conditions. Choosing the appropriate size of the bin width and the resulting amount of smoothing is a difficult problem in density estimation. Regarding histograms, setting *h* to approximately the square root of the number of observations (eq. 4.6) is a commonly used rule of thumb (Openshaw, 1995).

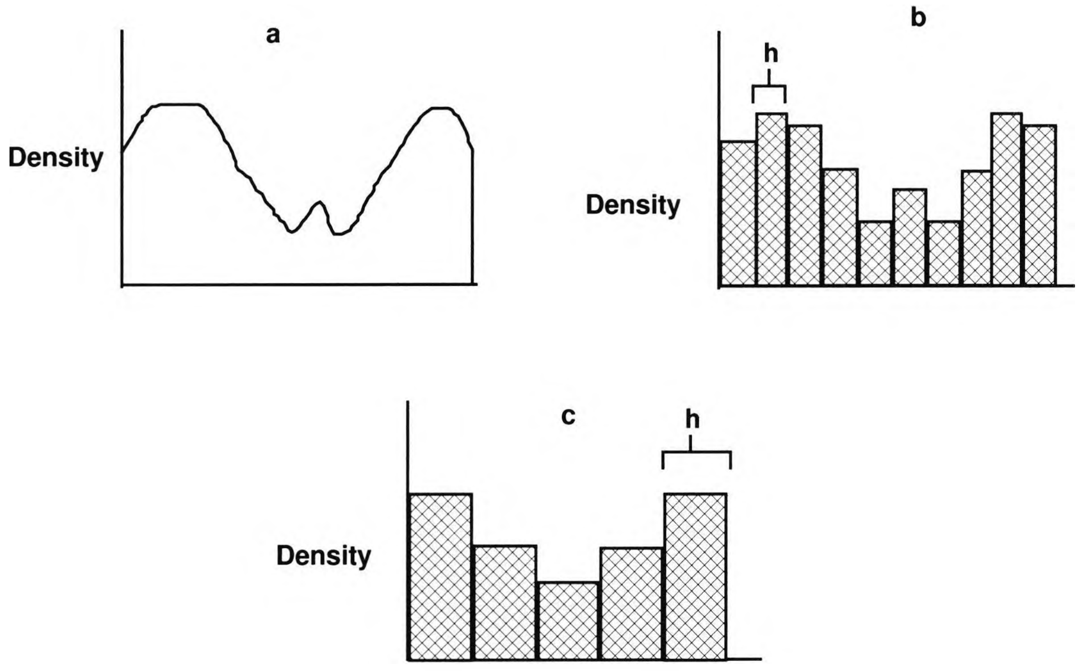


Figure 4.6: Increasing the bin size smoothes the density estimate.

For multivariate data histograms can be problematic as one must choose both the origin of the histogram and the orientation of the bins (Silverman, 1986). This is not a problem in APC because the origin is defined by one of the two cluster centroids under examination while the bins are defined by "slicing" the hypercylinder extending between the two centroids (as in Figure 4.4 and Figure 4.5).

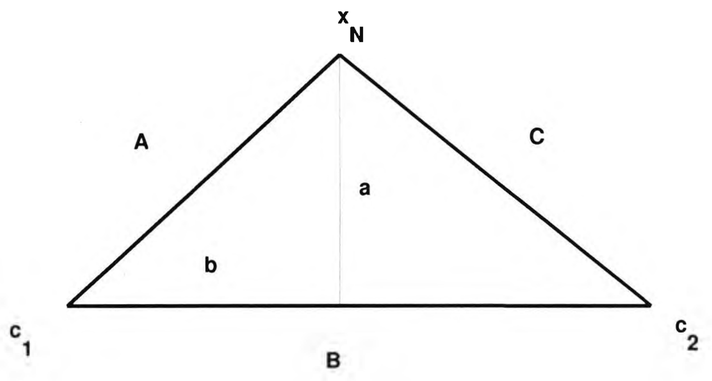


Figure 4.7: Points within the hypercylinder and the corresponding bin are found with the aid of the triangle formed by the two centroids and the observation.

Assuming one uses the rule of thumb given in eq. (4.6) for determining s_L and using equations 4.4 and 4.5 for s_W, using a histogram approach to measuring intercluster distance requires only one or two passes through the data set depending how one implements the histogram generating procedure. Regardless of the implementation three basic steps are required. For each observation, (1) determine if it falls within s_W of the line segment connecting the two centroids $\mathbf{c}_1$ and $\mathbf{c}_2$ - *i.e.* determine if the pattern falls within the hypercylinder. If so, (2) project the observation on the line segment and store its distance from one of the two centroids being used as a reference point. Given this distance, (3) calculate the bin in which the pattern belongs to. These steps can be accomplished as follows: referring to Figure 4.7, let $\mathbf{c}_1$ be the centroid of cluster 1 and $\mathbf{c}_2$be the centroid of cluster 2 and $\mathbf{x}_N$ be any observation in the data set. Let

$$\mathrm{A} = d(\mathbf{c}_1, \mathbf{x}_N) \qquad 4.8$$

$$\mathrm{B} = d(\mathbf{c}_1, \mathbf{c}_2) \qquad 4.9$$

$$\mathrm{C} = d(\mathbf{c}_2, \mathbf{x}_N) \qquad 4.10$$

where $d(\mathbf{w},\mathbf{y})$ is the distance (*i.e.* Euclidean) between objects **x** and **y**.

First, each observation in the data set must be tested to see if its projection onto the subsegment between $\mathbf{c}_1$ and $\mathbf{c}_2$ is orthogonal to the line segment. This is true if the following relation is true:

$$\frac{\left|A^2 - C^2\right|}{B^2} \leq 1.0 \qquad 4.11$$

If eq. (4.11) is true, then the distance, from $\mathbf{x}_n$ to the line segment can be calculated as:

$$\mathrm{a} = \left[A^2 - \left(\frac{A^2 + B^2 - C^2}{2B}\right)^2\right]^{1/2} \qquad 4.12$$

if eq. (4.11) is true and

$$a <= \mathrm{s_w} \qquad 4.13$$

is satisfied, the bin membership of $\mathbf{x}_n$ can be calculated from its distance, b, from $\mathbf{c}_l$:

$$b = (A^2 - a^2)^{1/2} \qquad 4.14$$

and a counter for that bin is increased. Once all patterns that fall within the hypercylinder have been allocated a bin, ADD and/or UDD can be calculated by taking at the largest bin in the case of ADD or, in the case of UDD, both the largest and smallest bins.

4.4 The Agglomeration Process and the Representation of Clusters

The agglomeration process generates a connected graph of the centroids found in the initial pattern partitioning stage. Links between centroids (*i.e.* nodes of the graph) are added in order of the centroids' distance (UDD or ADD) from each other (smallest to largest). This process will be illustrated via Figure 4.8 which shows a clustering problem with the initial centroids of the pattern partitioning stage already calculated. Figure 4.9 displays the distance matrices using ADD and UDD and Figure 4.10 is the resulting dendrograms. ADD first agglomerated centroids in cluster A while UDD first agglomerates the centroids in cluster B. Cluster B is agglomerated first by UDD because its density is more uniform. Note that the agglomeration is based on the initial centroids. At no point is the distance calculated between a centroid and an agglomerated cluster or between two agglomerated clusters. In this sense it can be seen that APC generates a connect graph of the initial centroids. The resulting dendrograms show at what point individual centroids agglomerated together. Each level of the dendrogram is the equivalent of looking at what would happen if links in the graph whose distance or weight greater than the current size are removed. Also note that infinite distance was found for all agglomerations between clusters *A* and *B*. This is because there are regions of zero density between the cluster centroids.

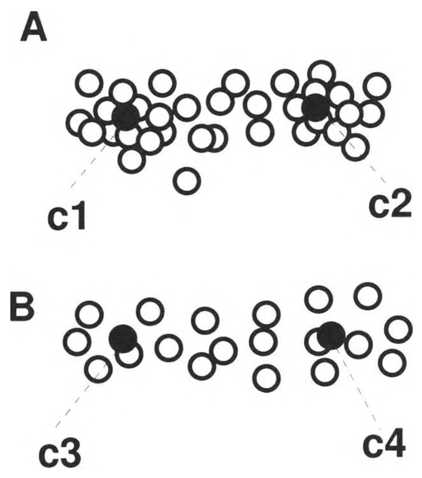


Figure 4.8: A two cluster, $k = 4$ clustering problem.

ADD

	c1	c2	c3	c4
c1	*	*	*	*
c2	5	*	*	*
c3	inf	inf	*	*
c4	inf	inf	3	*

UDD

	c1	c2	c3	c4
c1	*	*	*	*
c2	0.4	*	*	*
c3	inf	inf	*	*
c4	inf	inf	0.9	*

Figure 4.9: Distance matrices generated by ADD and UDD for the problem in Figure 4.8.

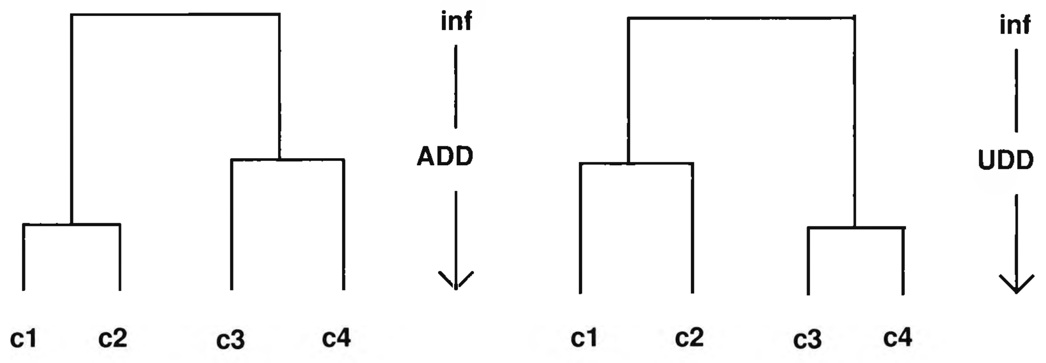


Figure 4.10: Dendrograms constructed from Figure 4.9.

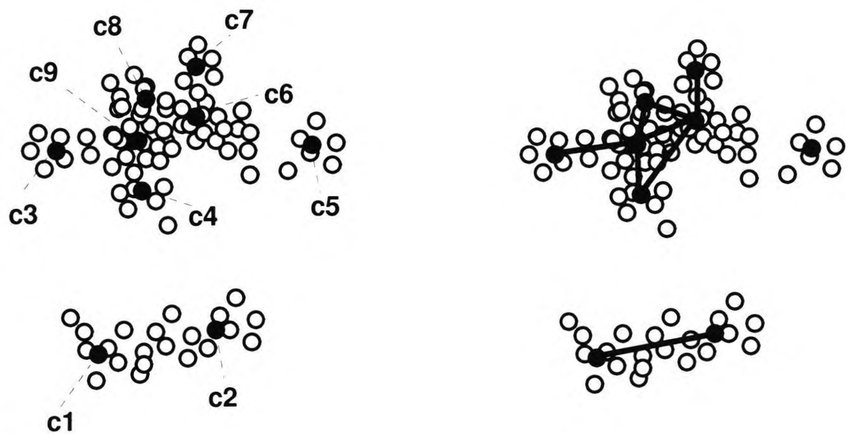


Figure 4.11: A three cluster problem and the APC representation of cluster structure. The heavy dots are the centroids of the initial clusters found in the pattern partitioning stage and the heavy lines indicate agglomerations between initial clusters. Note that the agglomeration between clusters can occur at any time between any two clusters that have not already been agglomerated together even if the two particular initial clusters have already been agglomerated into the same cluster but not with each other.

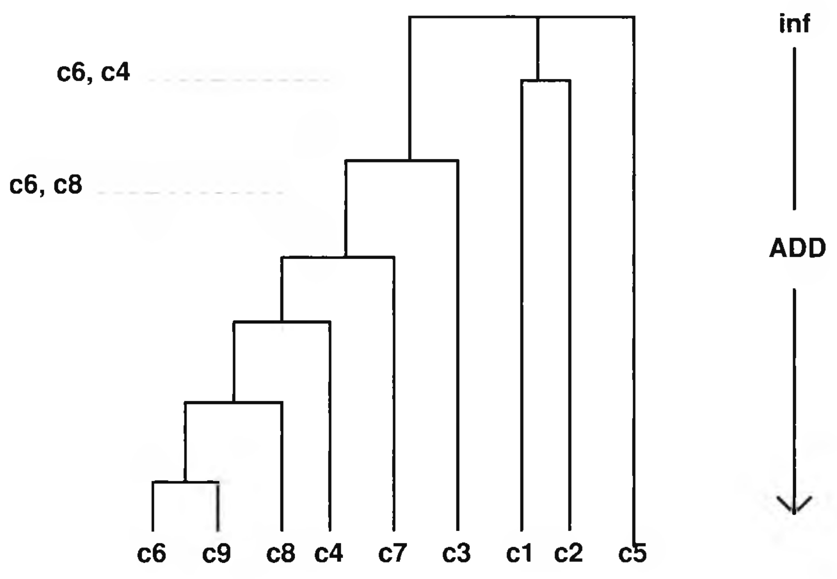


Figure 4.12: Dendrogram generated for Figure 4.11 by ADD. Note the dashed lines indicating agglomerations between initial cluster centroid that have already been agglomerated into the same cluster.

The agglomeration process is continued until all centroids are linked and thus the entire data set is modelled as a single cluster or all the remaining distances have zero density regions between them (infinite distance). While this process is going on, links are allowed to be established between centroids that are already agglomerated. This allows the linked line segments to span areas of high density as in the circle in Figure 4.1 or in Figure 4.11 above. If this were not allowed, the circle in Figure 4.1 would have a discontinuity in it and fail to take into account the fact that the cluster is a circle. Because of this, some additional information is added to the dendrogram. Figure 4.12 is the dendrogram of the cluster structure in Figure 4.11. An extra sub dendrogram (indicated by dashed lines) is added that indicates additional links (agglomerations) between centroids that already exist in the same agglomerated cluster. When a cut-off point is finally decided on, these extra links are left in place. If the cut-off is above the level at which the centroids are agglomerated the resulting *intracluster* links are used for measuring the cluster memberships of individual patterns.

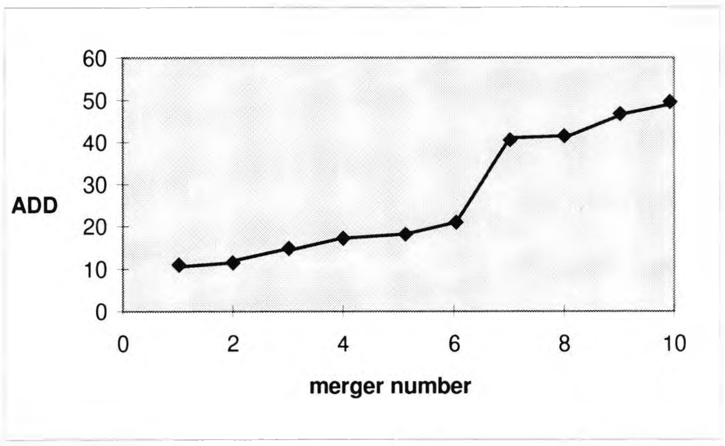


Figure 4.13: Plot of the ADD value at which clusters are agglomerated. The "elbow" at the 7th merger may be taken as a candidate point for stopping the agglomeration process and partitioning the data.

No "stopping rule" for finding the cut-off point specific to APC is being proposed in this thesis. However, a number of methods exist that are often used with other hierarchical clustering methods may be useful. For example, a simple and intuitive approach is to look at elbow plots of the distances at which clusters are merged. A sudden increase in the distances at which clusters agglomerate would be indicative of low density areas surrounding high density areas. Figure 4.13 shows a hypothetical elbow plot where at the 7th merger between clusters there is a sudden increase in the intercluster density as measured by ADD. Alternatively, along the lines of the MST approach to clustering discussed in Chapter 2, the succession of agglomerations can be treated as a connected graph and inconsistent edges that are some number of standard deviations greater than the edges in the subgraphs on either side can be used as cut-off points. The ratio of the size of the edge to the average on either side could also be used. In real world problems deciding on cut-off points will often not be strait forward or simple and so it may well be best to rely on a significant degree of domain knowledge if possible. Also, as mentioned earlier, it could be useful in this situation to generate two dendrograms one using ADD and the other using UDD and then referencing both dendrograms in deciding on a cut off point. A wide range of other stopping rules used in standard hierarchical methods are surveyed in Milligan and Cooper (1985). Wong (1982) has also proposed a stopping rule specific to WHM that could be used.

4.5 The Computational Cost of APC

The computational cost of APC can be broken down into three components: initial pattern partitioning, measuring intercluster distance and agglomeration.

4.5.1 Initial pattern partitioning

Assuming the use of k-means or the moving method, the computational cost of the pattern partitioning stage is $O(kN)$ where k is the number of initial clusters and N is the number of patterns in the data set. Note however that both k-means and the moving method are iterative algorithms each requiring a number of passes through the data set to converge taking $O(kN)$ time per pass.

4.5.2 Measuring intercluster distance

The first step in measuring intercluster distance is the determination of which clusters are neighbours. This process requires comparisons of the position of all k initial clusters which can be done in $O(k(k - 1)/2)$ time. Next, assuming the histogram approach described above is used for the intercluster density estimation, each of the N patterns must be tested to see if it falls within each hypercylinder. For each of the n_{ij} patterns that fall within the hypercylinder extending between clusters C_i and C_j, the bin in which it belongs in must be found. This can be done by projecting the pattern onto the hypercylinder axis and measuring the pattern's distance from one of the end points. Given this distance, the bin membership can be calculated in n_{ij} time. As there will be $(n_{ij})^{1/2}$ bins in the histogram (assuming eq. (4.6) for determining the number of bins) the computational cost of the histogram density estimate is O(N + $n_{ij}(n_{ij})^{1/2}$) per hypercylinder.

4.5.3 Agglomeration

Finally, the agglomeration stage requires the construction of a k x k object matrix and at most $k(k-1)/2$ distance (density estimation) calculations. Note that the true figure will normally be less than $k(k-1)/2$ as the distance between centroids need only be calculated for neighbouring clusters. The total computational cost for APC is therefore at most $O(kN) + [k(k - 1)/2]*[O(N + n_{ij}*(n_{ij})^{1/2})] + O(k(k - 1))2$. Again, it should be emphasised that this is most likely an overestimate as the intercluster distance (the $[k(k - 1)/2]*[O(N + n_{ij}*(n_{ij})^{1/2})]$ component of the above figure) is only calculated between neighbouring clusters.

4.5.4 An Empirical Demonstration of the Computational Cost of APC

The actual CPU cost of APC will vary across different clustering problems as the number of neighbouring initial clusters and the number of patterns within each hypercylinder between neighbouring cluster depends on the nature of the structures in the data. In order to put the CPU cost of APC into perspective an empirical comparison study of the CPU time required for the moving method and the agglomeration stage of APC was run. The moving method is a good benchmark with which to judge the overall time cost of APC as it is one of the faster clustering algorithms (see Chapter 6). Therefore it provides a practical yardstick with which to measure the CPU cost of the density estimation and agglomeration stages of APC.

In the simulation, APC using the moving method as an initial partitioning method was run on uniformly random data varying the number of patterns, number of initial clusters and the dimensionality. The *3 × 3* factorial experiment examined data sets of size *4,000*, *8,000* and *16,000* patterns; 5, 10 and 20 initial clusters and 5, 10 and 20 dimensions (variables). Each cell of the simulation was run 10 times and the average time for the pattern partitioning (moving method) stage and the combined density estimation and agglomeration stages were recorded[12]. The results are displayed in Table 4.1.

	Moving method	**Agglomeration and density estimation**	**Total**
Patterns	time (in seconds)	time (in seconds)	
4000	18.3	9.2	27.5
8000	59.8	18.8	78.6
16000	187.2	38.2	225.4
Dimension			
5	21.4	8.3	29.7
10	70.7	20.5	91.2
20	173.2	37.3	210.5
Partitions			
5	31.8	3.3	35.1
10	79.0	13.4	92.4
20	154.5	49.5	204.0

Table 4.1: The CPU processing time of the moving method compared to the agglomeration stage of APC.

In short, the agglomeration and density estimation components of APC required less processing time than the moving method (an ANOVA test applied to the results found that all differences across rows in Table 4.1 were significantly different at $p < 0.01$). This indicates that at least on the random data examined here the total CPU

[12] This simulation was run on a 133 Mhz Pentium PC.

time cost of APC (pattern partitioning, density estimation and agglomeration) is only fractionally greater than the moving method itself. However, it should be pointed out that for very large values of *k*, the CPU costs of the agglomeration stage may increase more rapidly than for the moving method. The degree of increase, though, will depend in large part on the geometric proximity of the initial clusters to each other. The greater the number of neighbouring clusters, the greater the increase in CPU cost.

4.6 Summary

The flow chart in Figure 4.14 summarises the APC methodology. The first step is to generate an initial *k* cluster partition of data using whatever clustering algorithm the analyst chooses. For large data sets k-means or moving methods are probably best due to their efficiency, but there is no theoretical restriction on what type of clustering technique is used so long as it can identify high density clusters. The second step is to construct a distance matrix using ADD and/or UDD. Histograms are recommended for use at this stage because of their simplicity and efficiency. Both measures, ADD and UDD can be used on their own or as complementary measures to determine which initial clusters are to be agglomerated. The resulting distance matrix can be either be treated as a connected graph where inconsistent links are removed or converted into a dendrogram and cut off at some point to determine the cluster agglomerations. The final step is to represent agglomerated clusters with line segments linking the original centroids together. The membership of an individual observation to its cluster is measured as the distance the observation is from the closest line segment in the cluster. This is discussed further in Chapter 8.

The next chapter discusses the experimental methodology with which the clustering ability APC will be compared to other hybrid methods. It includes a justification of the Monte Carlo evaluation method used as well as a discussion of the data sets used in the comparison.

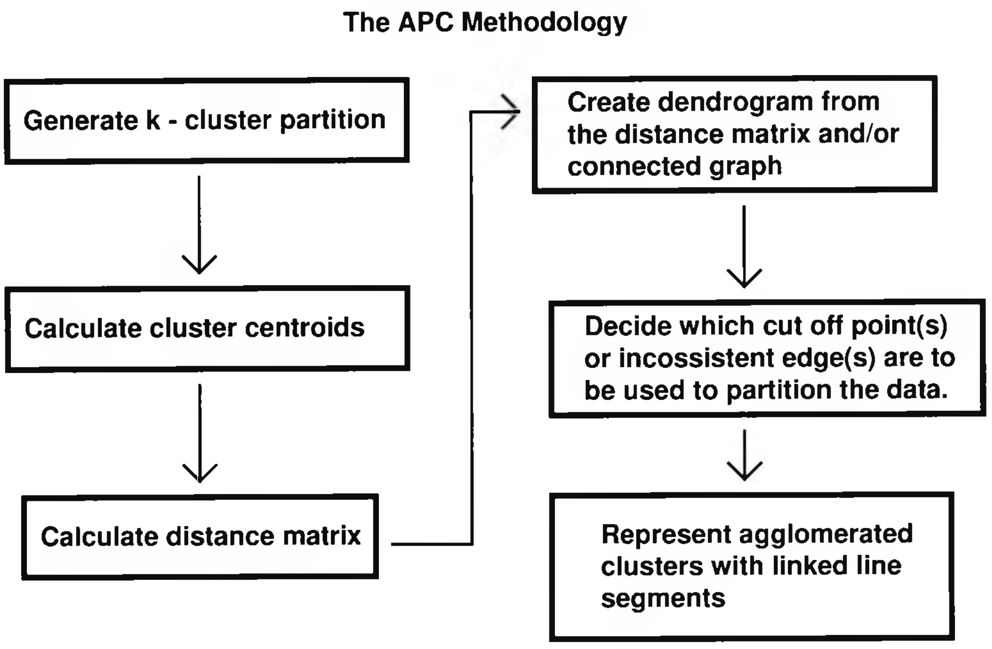

The APC Methodology
Generate k - cluster partition
Calculate cluster centroids
Calculate distance matrix
Create dendrogram from the distance matrix and/or connected graph
Decide which cut off point(s) or incossistent edge(s) are to be used to partition the data.
Represent agglomerated clusters with linked line segments

Figure 4.14: Flow chart of the APC methodology.

5. Methods: Monte Carlo Evaluation

Chapters 6 and 7 evaluate various clustering algorithms via Monte Carlo simulation. Chapter 6 compares a number of pattern partitioning algorithms for use in the initial clustering stage of APC while Chapter 7 compares the performance of APC to a number of other hybrid clustering methods. The purpose of this chapter is to give a general overview and justification of the experimental methods used. First, the Monte Carlo approach to assessing the relative clustering abilities of clustering algorithms is introduced. Next, the use of simulated as opposed to "real world" data in comparing the clustering performances of various clustering algorithms is justified. Section 5.3 briefly examines the benefits of Monte Carlo evaluation over the use of real word data. Finally, Section 5.4 overviews the types of simulated data sets that will be used in the Monte Carlo evaluations.

5.1 Monte Carlo Evaluation

The application of Monte Carlo methods to evaluating clustering techniques can be summarised as a three step process: 1) For each combination of experimental factors (*e.g.* number and type of clusters, noise, sample size *etc.*) use a computer program to generate a data set with the relevant features. As the cluster structure of the data is known, 2) apply each clustering technique to the data and 3) measure the ability to each method to correctly recover the cluster structure (Milligan, 1996; Milligan and Cooper, 1987). The same data set can be generated repeatedly (with some random variation) to produce as many samples of the same experimental conditions as desired by the experimenter.

A variety of methods exist for measuring the degree of correct cluster recovery by the clustering techniques (see Milligan and Copper (1986) for review). The Adjusted Rand Index (ARI) developed by Hubert and Arabie (1985) is widely accepted as one of the best measures of relative clustering ability when the actual clustering of the data is known as is the case with simulated data (Milligan, 1996), Milligan and Cooper, 1987) and is the one chosen for all comparative clustering experiments in this work. ARI measures the degree to which the classification of each pair of observations by the clustering algorithm is consistent with their known cluster membership. An ARI score of 1.0 indicates perfect correspondence and an ARI score near 0 indicates chance agreement. A more detailed description of the Adjusted Rand Index can be found in Hubert and Arabie (1985).

5.2 Problems with Using 'Real World' data to Evaluate Clustering Methods

Chapters 6 and 7 rely entirely on simulated data to compare clustering algorithms. The omission of "real world" data was deliberate, as there are strong arguments within the clustering literature cautioning against the use of real data in the evaluation of clustering methods. Discussions of the problems with using real data when comparing clustering methods and related issues can be found in Milligan (1996), Openshaw (1995), Milligan and Cooper (1987) and Dubes and Jain (1976). Salzburg (1999) also gives good recommendations on the use of real data relevant to the evaluation of both supervised and unsupervised classification methods.

The arguments against using real world data in comparison studies of clustering methods largely surround questions of how one measures the relative clustering abilities of clustering techniques on such data sets. Clustering methods are generally used as *ad hoc* data exploratory tools. The process of analysing and labelling the clusters resulting from the application of a clustering method to real world data is a subjective process that is at best problem dependent. Unlike supervised methods, there is no dependent variable which can be used to objectively assess the different methodologies. Given this, matching clusters across clustering methods on real world data is difficult to do empirically - if at all possible (Openshaw, 1995). Milligan (1996) and Milligan and Copper (1987) also point out that when this is attempted, the validity of the results can be questionable as one must have faith in the experimenter's *a priori* grouping of the data and assume that the clusters found by a particular clustering method are in fact appropriate or "correct". Furthermore, if a given clustering method fails to find the *a priori* clusterings, it is difficult to know if this is due to the clustering method, problems with the experimenter's *a priori* clusterings or due to there not actually being any structure in the data at all. The exception to this is when one is evaluating clustering techniques for use within a particular problem domain whose relevant cluster structure is well defined within a functional context. However, any comparisons between clustering methods may not generalise beyond that particular context.

One common approach to circumventing the subjectivity problem is to compare how well the clustering methods converge on the data. This approach is has been used to compare clustering methods that differ in how they optimise a given objective function (*e.g.* Choi and Park, 1994; Babu and Murty, 1993; Zhang and Boyle, 1991; Ismail and Kamel, 1989). For example, quality of convergence is often used in comparing variations of k-means which optimise the total within cluster squared error (WCSE). The problem with this is that a minimal WCSE does not necessarily lead to better clustering of the data even though minimising the WCSE is what these methods do (Openshaw, 1995; Nour and Madey, 1996). This is particularly likely to be the situation when clusters are not hyperspherical in shape.

Another approach is to use the clustering method as a classifier as one would a supervised method. However, unlike supervised classifiers, the clustering method is not given any information about the correct classifications of the objects during training. Once the data has been modelled by the clustering method, the "classification error" is measured based on the resulting clustering model's ability to correctly classify the data using the previously unseen grouping variable (*i.e.* the dependent variable). This makes little sense as one would generally not use a clustering technique in this way. Unless one is using the clustering method as a feature extraction method (see Chapter 8) and then inputting the results of the clustering method through supervised techniques, this gives little information about the relative clustering abilities of the methods under investigation.

5.3 The Advantages of Monte Carlo Evaluation

The Monte Carlo approach to validating clustering methods has a number of advantages over the use of real world data (Milligan, 1996). First, as the data is computer generated the experimenter can control the cluster structure of the data. Therefore, the true clustering of the data is known and the degree to which various clustering methods recover that structure can be precisely measured. Second, the ability of different methods to recover cluster structure under different conditions can be systematically evaluated. For example, if one is interested in the performance of clustering methods under different types of noise this can be easily tested by adding the relevant types of noise to the data. Finally, Monte Carlo simulation overcomes the difficulties with small sample sizes which is sometimes a problem with real world data as the same cluster structure in the data can easily be replicated (with random variation) and tested many times. For example, in this work each data set was recreated and tested 30 times.

It should be pointed out however, that the main disadvantage of Monte Carlo simulation is that the relative clustering abilities of the methods being examined may not generalise beyond the actual cluster structures or data characteristics examined (Milligan, 1996). In many natural data sets there may exist characteristics in the data that may not have been incorporated into the Monte Carlo simulation. Therefore, replication of simulations and the continued expansion of the types of cluster structure used to examine clustering techniques are important.

5.4 Data Sets

As APC is designed to recover both hyperellipsoidal and nonhyperellipsoidal cluster shapes, the experiments in Chapters 6 and 7 test the effects of cluster shape on the ability of various clustering algorithms to recover cluster structure. It should be emphasised at this point that what is really being examined is not the ability of these methods to recover specific cluster shapes, but rather how well they recover a few basic categories of cluster shape. In other words, cluster shape is being used as a general experimental design factor. As there are an infinite number of cluster shapes of which each can be infinitely varied, testing the recovery abilities on all types of cluster shape is obviously not feasible. The purpose of using cluster shape as an experimental design factor in this thesis is to demonstrate the greater flexibility of APC regarding the recovery of nonhyperellipsoidal cluster shapes relative to other hybrid clustering methods.

5.4.1 Cluster Shapes Used in the Literature

In any case, the use of cluster shape as a design factor in Monte Carlo evaluations of clustering methods has not received much attention. This is probably due to the fact that most Monte Carlo studies come from the social sciences literature (particularly behavioural sciences) where assumptions of multivariate normality in data are common place (Balakrishnan *et al,* 1994). Table 5.1 displays a sample survey of cluster shapes that have been used in Monte Carlo studies. Most of these studies were comparing the relative performances of clustering methods although some were examining other issues related to cluster analysis such as stopping or cut-off rules as well as various performance measures. 19 of the 24 studies examined only investigated multivariate normal or truncated multivariate normal mixtures. More importantly, of the 24, only Helmstadter (1957) measured cluster recovery while systematically varying the types of cluster shape. As will be shown in the next chapter, cluster shape can significantly effect the relative performances of clustering algorithms.

Unconstrained Multivariate Normal Clusters	**Truncated Multivariate Normal Clusters**	**Miscellaneous Cluster Types**
Celeux and Govaert (1995)	Balakrishnan *et. al.* (1996)	Blashfield and Morey (1980) Artificial psychiatric data
Overall and Magee (1992)	Mangiameli *et al* (1996)	Milligan and Isaac (1980) Ultrametric
Wharton (1984)	Chen *et. al.* (1995)	Mojena (1977) Multivariate Gamma
Edelbrock (1979)	Balakrishnan *et. al.* (1994)	Mezzich (1978) artificial psychiatric data
Bayne *et al* (1980) (Bivariate)	Milligan and Cooper (1986)	Helmstadter (1957) Geometric Shapes
Blashfield (1976)	Scheibler and Schnieder (1985)	
Kuiper and Fisher (1975) multivariate normal and bivariate	Milligan and Cooper (1985)	
Gross (1972)	Milligan *et al* (1983)	
Rand (1971)	Milligan (1981b)	
	Milligan (1980)	

Table 5.1: Sample survey of Monte Carlo evaluations in the clustering literature.

5.4.2 Cluster Shapes Used in this Thesis

This thesis limits itself to 4 basic categories of cluster shapes: multivariate normal hyperellipsoids, chaining clusters, chaining clusters with variable point density (cones) and a mixed condition where all three cluster shapes are present (Figure 5.1). In addition, other experiments test the effects of placing chaining and hyperellipsoidal clusters in close proximity to each other (Figure 5.2). Other experimental factors investigated along with cluster shape in later chapters include level and type of noise, dispersion of the clusters, number of clusters and dimensionality of the data. Details of how the cluster shapes were generated as well as how the other experimental factors were incorporated into the data sets can be found in the "methods" sections of the relevant chapters and in Appendix A3 which contains sample code of the data generation routines used in this thesis.

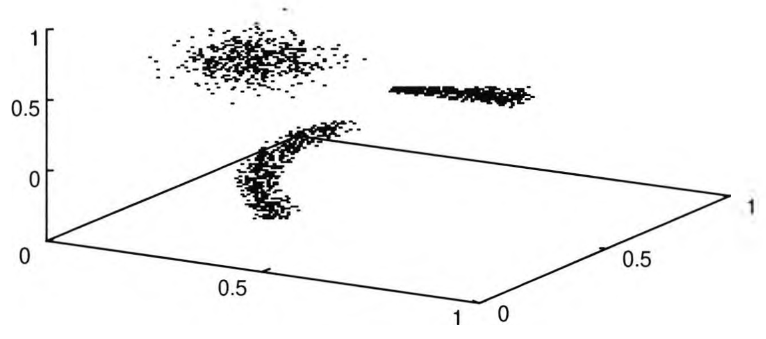


Figure 5.1: Cluster shapes: hyperellipsoid, arc and cone used in this evaluation (Chapters 6 and 7).

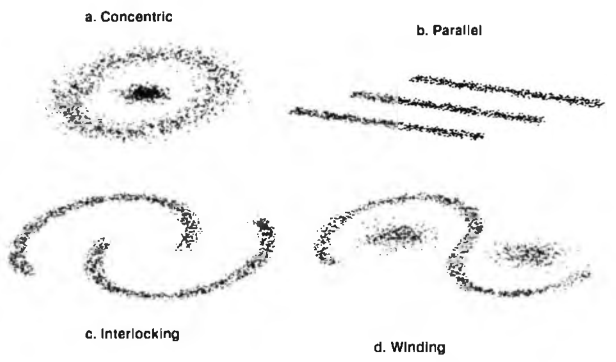


Figure 5.2: Close proximity chaining and hyperellipsoidal clusters used (Chapters 7 and 8).

5.5 Summary of Experiments

Chapter 6 demonstrates how cluster shape can effect the relative performances of different clustering algorithms. In this chapter, k-means, the moving method and two self-organising neural networks are evaluated via a Monte Carlo simulation using the cluster shapes in Figure 5.1. In addition to demonstrating the effects of cluster shape, this chapter empirically demonstrates that the moving method consistently converges faster that k-means over a wide variety of conditions and argues that this method is superior to k-means , WTANN and KSONN as an initial partitioning algorithm in APC.

Chapter 7 presents the actual comparisons between APC and six other hybrid clustering methods. Monte Carlo evaluations are run on both sets of clusters shapes (Figure 5.1 and Figure 5.2). The effects of noise, dimensionality, cluster dispersion, number of initial partitions are examined. In addition, varying the number of known clusters in the data is also investigated using the Figure 5.1 data.

Finally, Chapter 8 empirically demonstrates the benefits of using linked line segments over centroids and principal component scores to model cluster shape. APC, k-means and principal component analysis (PCA) are applied to three of the data sets in Figure 5.2 (winding, concentric and interlocking). The cluster shapes are intrinsically two dimensional but in this experiment they were imbedded in 10 dimensional space. The above three data reduction approaches were then used to try and reduce the dimensionality of the data while faithfully maintaining the distinction between the clusters in each data set. This was tested by taking the reduced data as produced by APC, k-means and PCA and presenting it to a linear discriminant model to see how well it could use the reduced data to distinguish between the data structures.

6. A Monte Carlo Evaluation of Four Pattern Partitioning Techniques

The main objectives of this chapter is to 1) evaluate k-means, the moving method and two types of self-organised neural networks (SONN's) as methods for generating the initial pattern partitioning in APC and 2) demonstrate that cluster shape can effect the relative performances of clustering algorithms. Interest in the comparative clustering abilities of k-means, moving methods and self-organising neural networks has been increasing in recent years. However, most comparative studies have either been restricted to specific problem areas or have been conducted under other limitations that do not provide a more general evaluation of the relative abilities of these methods under a wide variety of conditions. This chapter provides a systematic empirical evaluation of the clustering abilities of k-means, moving methods and the two types of self organising neural networks discussed in Chapter 3 (WTANN and KSONN). The effects of cluster shape, dimensionality, noise, dispersion and number of clusters in the data is used to evaluate the above methods via Monte Carlo simulation. Results indicate that under most conditions k-means, the moving method and WTANN perform equally well in terms of clustering ability. However, as the moving method consistently converges faster than k-means, it may well represent a more appropriate benchmark for future comparisons between pattern partitioning methods. Moreover, the results also demonstrate that cluster shape can effect the relative performances of different clustering algorithms.

6.1 Introduction

In recent years there has been an increasing amount of attention paid to the use of Kohonen self-organising neural networks and related "winner take all" self-organising networks in cluster analysis. A number of studies have examined the similarities between Kohonen self-organising neural networks (KSONN), winner take all self-organising neural networks (WTANN) and the k-means algorithm (*e.g.* Chinrungrueng and Sèquin, 1995; Chen and Titteringdon, 1994; Ripley, 1992) and some have included empirical comparisons (*e.g.* Balakrishnan *et al*, 1996, 1994; Openshaw, 1995; Murtagh and Hernández-Pajares, 1992). In general, the empirical results seem to indicate that although k-means often performs slightly better, these methods are roughly comparable. However, most of these studies were either confined to specific problem domain areas or were not very exhaustive in terms of the types of cluster shapes examined.

Although k-means seems to be the accepted benchmark in comparing pattern partitioning methods such as KSONN and WTANN (*e.g.* Balakrishnan *et al*, 1996, 1994), work by Ismail and Kamel (1989) and Zhang and Boyle (1991) have suggested the moving method (Duda and Hart (1973), Ismail and Kamel (1989)) often outperforms k-means both in terms of speed and quality of convergence without additional computational cost. However, both of these studies were conducted using a small number of two dimensional empirical data sets. Given the limited range of data that k-means and the moving method were evaluated over, a more systematic investigation of the relative performances of the two partitioning methods is needed.

The objective of this chapter, therefore, is to expand on a number of the above comparison studies investigating the relative clustering abilities of k-means, the moving method, WTANN and KSONN. Unlike previous comparisons, this evaluation covers a range of cluster structures as well as investigating the effects of noise, dimensionality, cluster dispersion and the number of clusters present in the data. The first two studies of interest are Ismail and Kamel (1989) and Zhang and Boyle (1991) who compared the performance of k-means and the moving method. Zhang and Boyle showed that although the moving method possesses all of the convergence states of k-means, not all moving method convergence states are accessible to k-means. If any of these additional convergence states represent better clusterings of the data, this could lead to better performance with the moving method (see Chapter 3). However, the opposite could also be true - the additional convergence states could possibly increase the likelihood of the moving method to get stuck in local minima. Both Ismail and Kamel and Zhang and Boyle ran empirical comparisons of the two techniques and found that the moving method often does converge to better optima than k-means, and in the case of Zhang and Boyle, with significantly fewer iterations through the data set. However, between these two studies the comparisons were conducted over only 5, two dimensional data sets. A more systematic and controlled comparison of the two techniques under a wider variety of conditions is needed.

Also, their empirical comparison measured the relative clustering abilities based on the sum of the within cluster error:

$$E = \sum_{j=1}^{k} \sum_{i \in C_j} \left\| \mathbf{c}_j - \mathbf{x}_i \right\|^2 \qquad 6.1$$

where

$$\mathbf{c}_j = \frac{1}{n_j}\sum_{i \in C_j} \mathbf{x}_i \qquad 6.2$$

and n_j is the number of observations in cluster C_j and $\| \;\|$ is the Euclidean norm, as opposed to a more direct metric of classification performance such as the Adjusted Rand Index. Although comparing performance on eq. (6.1) is logical in the sense that both methods explicitly minimise eq. (6.1) and that both studies relied primarily on "real world" empirical data where the actual clusterings may not have been known, the optimisation of eq. (6.1) does not necessarily coincide with optimal recovery of cluster structure (Nour and Madey, 1996; Openshaw, 1995). Alternatively, the Adjusted Rand Index (ARI) gives a more direct and clear indication of cluster recovery. In addition, it is widely accepted as the best measure of relative clustering ability when the actual clustering of the data is known as is the case with simulated data (Milligan, 1996; Milligan and Cooper, 1987).

The third study to be examined here is Balakrishnan *et al* (1994) who systematically compared k-means, KSONN and WTANN in a Monte Carlo evaluation. The conclusion of their work was that k-means generally outperforms both KSONN and WTANN. However, they pointed out that as their research was intended for the behavioural sciences, this led to a couple limitations worthy of future research. First, they only examined data sets with 50 observations on the grounds that sample sizes on this order are quite common in the behavioural sciences. They suggest that this may have biased the results towards k-means as both KSONN and WTANN may perform better when there is ample data to allow "fine tuning" of the clusters (Kangas *et al*, 1990). On larger data sets, the relative abilities of the two methods might be different. Second, they only examined truncated hyperellipsoidal cluster structures following a data generation method developed by Milligan (1985). Again, they felt that this was appropriate for the behavioural sciences where this type of structure is common. However, they suggest that the use of hyperellipsoidal clusters may have also biased the study in favour of k-means and that more comparative studies involving other cluster structures is needed.

Building on Balakrishnan *et al* (1996, 1994), Zhang and Boyle (1991) and Ismail and Kamel (1989) this chapter will evaluate the above clustering methods using a large sample of data as well as using other cluster structures in addition to hyperellipsoids. In short, this chapter evaluates k-means, the moving method, KSONN and WTANN via Monte Carlo simulation under a variety of conditions including cluster shape, dimensionality, cluster dispersion, noise and number of clusters. As simulated data is used, cluster recovery can be measured directly via ARI. A relatively large sample size of 1500 data patterns per data set is used to reduce any sample size bias in favour of k-means and the moving method. In addition, the convergence speed of both k-means and the moving method are evaluated over all conditions and data sets.

The remainder of this chapter is divided into five sections. Section 6.2 outlines the Monte Carlo methodology used in the evaluation and discusses how the test data was generated. The next two sections present the results of the Monte Carlo simulation which are analysed both in terms of relative clustering ability and speed of convergence. Finally, section 6.5 provides some concluding remarks and some suggestions for future work.

6.2 Methods

The k-means method used is the Forgy algorithm described in Chapter 3. For the moving method, all potential cluster movements were tested for each pattern to find the best possible move (*i.e.* that which produces the greatest reduction in eq (6.1)). Both k-means and the moving method were seeded using random patterns. Both KSONN and WTANN were run with one dimensional output layers with the number of units in the output layer were set equal to the number of known clusters in the data set. For the KSONN the neighbourhood function was initialised with r = 2/3 of the number of units in the output layer and reduced linearly during training. For both networks the learning rate (η) was initialised at 0.2 and also reduced linearly during training. Both networks were trained for 1000 epochs where one epoch is a complete pass through the entire data set.

6.2.1 Data generation

The majority of previous Monte Carlo studies have concentrated on the comparison of clustering methods using multivariate normal clusters (see Chapter 5). In this study cluster shapes in addition to multivariate normal hyperellipsoids were evaluated. These additional shapes were arcs and cones. The purpose of the additional cluster shapes was to evaluate the performance of the four clustering methods on non "globular" or hyperellipsoidal cluster structures. The arc cluster shape was intended to test the performance of the various methods on the ability to handle chaining structures. The third condition involved the use of cones whose point density was varied along the axis. This was meant to examine the ability of the methods ability to cluster chaining clusters with varying point densities. Finally, a fourth condition was added that included all of the cluster shapes in the data set simultaneously.

The data sets were generated to comply with the concepts of internal cohesion and external isolation (Cormack, 1971). Internal cohesion refers to similar patterns occupying the same area of the pattern space. External isolation means that individual clusters are isolated from other clusters. The hyperellipsoid clusters were generated by placing normally distributed points with variance = σ and mean = g_d in each dimension, d, around a fixed point in space, $\mathbf{g} = \{g_1, g_2, \dots, g_d\}$. No truncation was used as in many other Monte Carlo studies (see Milligan and Cooper, 1987) as it was felt that a multivariate normal distribution of patterns is more natural. The conic clusters were created by generating a straight line (axis) of length $l = 0.28$ with midpoint specified by **g**. Although somewhat arbitrary, the value 0.28 was chosen so that the conic clusters would have an overall "elongated" structure but short enough to reduce the probability of the clusters intersecting. The direction of the uniformly distributed density gradient was randomly chosen to begin at one of the ends of the axis. The points at the beginning end were placed within a radius of 12.5% of the dispersion, σ, (see below) from the axis which was increased linearly towards the other end of the line whose radius was set to 50% of σ. The arcs were created by generating a 120° arc in one, randomly chosen dimension of radius 0.2 from a specified centre **g**. For each additional dimension, a uniform distribution of points within a radius of 50% of σ from the arc was placed to create a 2 dimensional arc in d-dimensional Euclidean space. A diagram of the cluster shapes used can be seen in Figure 6.1.

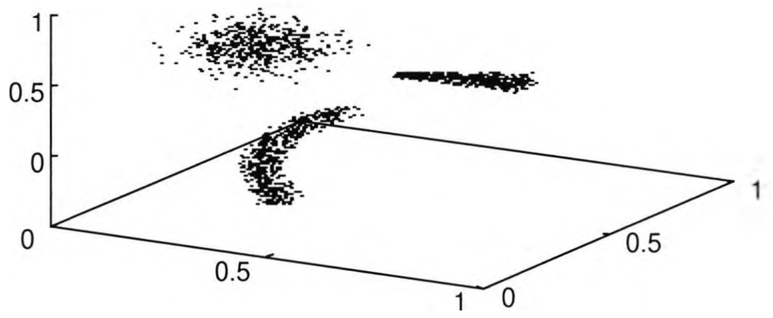


Figure 6.1: Cluster shapes: hyperellipsoid, arc and cone used in this evaluation.

In order to help ensure a degree of cluster separation, the following procedure was used in determining the "centre" **g**, from which each of the clusters was constructed. Let k be the number of clusters. For each dimension, d, the axis in the range of 0 to 1 was divided into $k + 1$ parts to determine the k most separated points. The k centres were then assigned one of these points for each dimension. However, for each dimension, it was ensured that the order of in which the k points was assigned to each of the $\mathbf{g}$'s was changed. For example, for two cluster centres, $\mathbf{g}_1$ and $\mathbf{g}_2$ in two dimensions, the resulting centres would be: $\mathbf{g}_1 = g_{k1}, g_{k2}$ and $\mathbf{g}_2 = g_{k2}, g_{k1}$ where k_1 is the closest of k points to the origin of axis d and k_2 is the second closest. The orientations of the cones and arcs were chosen randomly. In the mixed condition, cluster shapes were also randomly chosen. A more detailed discussion of the data generation along with sample code can be found in Appendix A3.

In addition to cluster shape, four other factors were incorporated into the Monte Carlo simulation. The number of clusters was varied from 3, 5 and 7. Cluster dispersion was varied from $\sigma = 0.025$, 0.05 and 0.075. This range was chosen to be large enough to effect the size of the clusters without obliterating the cluster shapes or causing an excessive degree of cluster overlap. 5 types of noise were also examined. Level 0 was no noise present in the data. Levels 1 and 2 had σ and 2σ Gaussian noise added to 50% of the patterns. The Gaussian noise is the same as that recommended by Milligan (1985) and is effectively a simulation of measurement error. Level 3 noise was 25% of the patterns replaced with uniform noise in the range of the data set. Level 4 noise had 50% of the patterns replaced with uniform noise. This type of noise simulates uncorrelated structure in the data as was used following the example of Cowgill (1993). The dimensionality of the data was also varied from 4, 7 and 10 dimensions.

In total, the Monte Carlo simulation was a 4 (cluster shape) x 5 (noise) x 3 (dispersion) x 3 (dimension) x 3 (number of clusters) factorial model. The number of data patterns (N) was fixed at 1500. For each of the 540 cells, 30 replications were run for a total of 16,200 data sets generated. The relatively large number of replications was intended to help average out any unusually good or bad performances due to the seeding procedure used. Clustering performance was measured using ARI.

6.3 Results

The results were analysed via a series of analyses of variance using the ANOVA procedure of SAS[13]. In addition to comparing the overall performance of the four clustering algorithms, the effects of each factor on each clustering algorithm as well as the relative performance across clustering models on each level of each design factor was examined.

Table 6.1 displays the mean ARI score for each clustering method and the mean number of iterations for k-means and the moving method. K-means, moving method and WTANN performed equally well (ARI = .582, .583 and .583 respectively) with no significant difference at the 0.01 level between ARI scores. Only KSONN recovered significantly less cluster structure than the others (ARI = .516).

	KM	**MV**	**WTANN**	**KSONN**
Mean ARI	.582*	.583*	.583*	.516
Mean Iterations	11.03	7.18	(1000)	(1000)

Table 6.1: Mean Adjusted Rand (ARI) scores for the four clustering methods. "*" indicates no significant difference at the 0.01 level. Also shown are the mean number of iterations for k-means and the moving method (both KSONN and WTANN were run for a fixed 1000 iterations) The difference in mean iterations between k-means and the moving method is significant at the 0.01 level.

The left hand sides of Table 6.2 and Table 6.3 display the mean ARI for each clustering method over each level of all factors. Table 6.2 displays the effect of each factor on each model (under each model, scores within the same factor with the same letter are not significantly different at the 0.01 level). All performed best on the

[13] Release 6.09.

hyperellipsoid clusters followed by the mixed, arc and cone. Increasing the number of clusters decreased cluster recovery for all methods although for k-means and the moving method there was no significant difference between 5 and 7 clusters and no significant difference between 3 and 5 clusters for WTANN. Increasing the dispersion significantly deceased performance for all methods except KSONN where it led to a significant increase in cluster recovery. Increasing the dimensionality significantly improved performance of all methods except KSONN while all methods showed roughly similar performances under the noise conditions.

	KM	**MM**	**WTANN**	**KSONN**		**KM**	**MM**	%
	ARI	ARI	ARI	ARI		Iterat'ns	Iterat'ns	Differ'ce
SET								
HyEllip	.665	.669	.742	.593		9.77	6.45 a	34%
Cone	.488	.488	.451	.415		13.10	8.39	36%
Arc	.574	.573	.486	.487		10.30	6.62 a	36%
Mixed	.601	.602	.654	.569		10.97	7.24	34%
CLUSTS								
3	.626	.628	.599 a	.625		8.19	5.13	37%
5	.557 a	.558 a	.596 a	.502		12.02	7.89	34%
7	.563 a	.563 a	.555	.421		12.90	8.51	34%
VARS								
4	.552	.553	.563	.521 a		11.25 a	7.15 a	36%
7	.576	.578	.577	.509		11.31 a	7.35a b	35%
10	.618	.619	.610	.519 a		10.55	7.02 b	34%
NOISE								
None	.795	.793	.796	.640		7.11	4.53	36%
Gauss 1	.763	.768	.772	.659 a		8.15	5.23	36%
Gauss 2	.728	.728	.731	.664 a		9.69	6.18	36%
U 25%	.436	.438	.430	.430		12.53	8.19	35%
U 50%	.187	.188	.189	.187		17.69	11.75	34%
DISP								
0.025	.604	.607	.602	.499		10.02	6.55	35%
0.05	.576	.579	.586	.523 a		11.29	7.33	35%
0.075	.566	.563	.562	.526 a		11.79	7.65	35%

Table 6.2: Mean ARI score and iterations by cluster shape, error, dispersion, dimension and number of clusters. ARI scores under the same model within a factor with the same letter (*e.g.* "a" or "b" are not significantly different at the 0.01 level.

Table 6.3 gives the relative performance of each model for each level of each factor. ARI scores in the same row with the same letter are not significantly different at the 0.01 level. The results in Table 6.3 indicate that there were no significant differences between the cluster recovery abilities of k-means or the moving methods under any level of any factor. Under the cluster shape condition, k-means and the moving method performed best with cone and arcs, relative to the neural networks while WTANN produced the best recovery under the hyperellipsoids and mixed conditions. KSONN produced the worst performance under hyperellipsoids, cones and mixed clusters and was not significantly better than WTANN under the arcs.

	KM	**MM**	**WTANN**	**KSONN**		**KM**	**MM**	%
	ARI	ARI	ARI	ARI		Iterat'ns	Iterat'ns	Differ'ce
SET								
HyEllip	.665 A	.669 A	.742	.593		9.77	6.45	34%
Cone	.488 A	.488 A	.451	.415		13.10	8.39	36%
Arc	.574 A	.573 A	.486 B	.487 B		10.30	6.62	36%
Mixed	.601 A	.602 A	.654	.569		10.97	7.24	34%
CLUSTS								
3	.626 A	.628 A	.599	.625 A		8.19	5.13	37%
5	.557 A	.558 A	.596	.502		12.02	7.89	34%
7	.563 A	.563 A	.555	.421		12.90	8.51	34%
VARS								
4	.552 A	.553 A	.563	.521		11.25	7.15	36%
7	.576 A	.578 A	.577 A	.509		11.31	7.35	35%
10	.618 A	.619 A	.610 A	.519		10.55	7.02	34%
NOISE								
None	.795 A	.793 A	.796 A	.640		7.11	4.53	36%
Gauss 1	.763 A	.768 A	.772 A	.659		8.15	5.23	36%
Gauss 2	.728 A	.728 A	.731 A	.664		9.69	6.18	36%
U 25%	.436AB	.438 A	.430 B	.430 B		12.53	8.19	35%
U 50%	.187 A	.188 A	.189 A	.187 A		17.69	11.75	34%
DISP								
0.025	.604 A	.607 A	.602 A	.499		10.02	6.55	35%
0.05	.576 B	.579AB	.586 A	.523		11.29	7.33	35%
0.075	.566 A	.563 A	.562 A	.526		11.79	7.65	35%

Table 6.3: Mean ARI score and iterations across clustering model by cluster shape, error, dispersion, dimension and number of clusters. ARI scores across models under the same factor level with the same letter (*e.g.* "A" or "B") are not significantly different at the 0.01 level.

Under the number of clusters, dimension, noise and dispersion conditions, both k-means and the moving method were among the best performing group of clustering algorithms except 5 clusters, 4 variables and 0.05 dispersion. WTANN performed equally well as either k-means or the moving method under all levels of these factors except 7 and 3 clusters while under 5 clusters and 4 variables it significantly outperformed both methods. KSONN performed significantly worse than the other three clustering techniques under all conditions except 3 clusters and uniform noise. Under the arc cluster shape its performance equalled WTANN.

Looking at the mean number of iterations through the data sets, (bottom row of Table 6.1 and right hand side of Table 6.2 and Table 6.3) the moving method converged an average of about 4 iterations faster than k-means (7.18 and 11.03 respectively), about 35% faster. The left hand side of Table 6.2 and Table 6.3 gives the mean number of iterations for k-means and the moving method for each level of each condition. First, it is immediately apparent that an ~35% faster convergence is significant and consistent across all levels of all conditions only ranging between 34% and 37%.

Table 6.2 and Table 6.3 also indicate the effects of each condition on the convergence rate of the two methods is similar. Both methods converged fastest on the hyperellipsoid data, followed by the arc, mixed and cone. Increasing the number of clusters increased the number of iterations required. Increasing the dispersion and to some extent the dimensionality also increased the converge rate.. The convergence speed of both methods were also similarly effected by the noise conditions.

6.4 Discussion

Glancing at Table 6.3, the most vivid result is that k-means and the moving method recover cluster structure equally well under all conditions examined here. Given

4+3+3+5+3 = 18 possible comparisons between clustering techniques, k-means and the moving method were both in the best performing group 13 times (72.2%), as was WTANN. KSONN was only in the best performing group twice (11%). Given this and the results in Table 6.1, on average k-means, the moving method and WTANN seem to perform equally well. The use of a neighbourhood function, however, does seem to impair the clustering ability of KSONN. This may well be due to the added constraint in this technique that similar clusters be placed in proximity in the output layer. It should also be pointed out, however, that KSONN is designed primarily for generating topological maps of the data as opposed to the type of clustering application explored here (Kangas *et al,* 1990).

Regarding cluster shape, the relative performance of all the methods was the same. All performed best on the hyperellipsoid clusters, next best in the mixed condition followed by the arc and cone. However, within each shape condition there was some variability suggesting that cluster shape can effect relative performance for both networks. K-means and the moving method produced nearly identical ARI scores on each cluster shape. Varying the shape of the cluster, at least those examined here, did not produce a bias for one of these two methods over another. However, the same is not true for the networks. WTANN recovered significantly more cluster structure in the hyperellipsoid and mixed conditions than any other method. This suggests the superior k-means performance found in Balakrishnan *et al* (1994) was probably due to sample size and not the hyperellipsoidal cluster structure used in their study. Although WTANN performed significantly better on the hyperellipsoids and the mixed condition, the cone and arc conditions gave it some difficulty as it performed much worse than k-means and the moving method. In fact, its performance on these conditions was more similar to that of KSONN.

Increasing the number of clusters decreased the clustering abilities of all methods. This effect was strongest for KSONN. WTANN was less effected by this than either k-means or the moving method. Increasing the number of clusters from 3 to 7 led to a 33% drop in KSONN performance. For k-means and moving method the drop was 10%. WTANN was only decreased by 7%. These results are similar to that in Balakrishnan *et al* (1994) who found that use of a neighbourhood function led to a larger increase in error rate with the increase in the number of clusters than with k-means or WTANN.

Increasing the dimensionality improved performance of all the methods except KSONN. This is probably due to the larger amount of redundant information in higher dimensions. K-means, moving methods and WTANN produced very similar ARI scores for each level suggesting they are similarly effected by dimensionality. The effect of increasing the number of variables from 4 to 7 initially deceased the KSONN performance however the increase from 7 to 10 improved it. In absolute terms though, the performance of KSONN was much less effected by changes in dimensionality than the other three methods.

Noise effected k-means, moving method and WTANN nearly identically. All performed at roughly the same level for each type of noise. Again KSONN is the exception: adding Gaussian noise actually *improved* cluster recovery. This is interesting as the addition of Gaussian noise increases the degree of cluster overlap. Increasing the dispersion (and cluster overlap) also improved the performance of KSONN while decreasing the performance of the other methods.

Given that increasing the dispersion and the Gaussian noise increase cluster overlap, this result suggests an interaction between overlap and the neighbourhood function. It could be that less well separated clusters make it easier for individual output units to change their cluster membership as the neighbourhood function approaches 0. When there are fewer patterns between clusters however, it is difficult for a given unit to establish itself in a well separated cluster away from neighbouring units because there are no intervening patterns to drag it away. In other words, it can get "stuck" in its neighbour's cluster. A similar result involving well separated clusters was found by Balakrishnan *et al* (1994) and in Mangiameli *et al* (1996) with networks using a neighbourhood function. One of the rationales of using a neighbourhood function is to help prevent individual units in the output layer from consistently "losing out" during the training process and not responding to any input patterns (Hertz *et al*, 1991). Perhaps the cost of using this approach is that the likelihood of output units responding to the same clusters within the data is increased.

Finally, although there is no difference in terms of cluster recovery between k-means and the moving method under any of the conditions investigated here, moving methods have been shown to consistently converge faster. A mean 35% increase in convergence speed was seen across all conditions. It seems that the possession of additional convergence states by the moving method improves converge speed as opposed to cluster recovery under the types of conditions examined here. This conflicts with the results of Ismail and Kamel (1989) and Zhang and Boyle (1991) who found that the moving method converged to lower minima than k-means. In fact, a *post hoc* analysis examining the mean value of eq. (6.1) at convergence for the two algorithms indicated that they still perform nearly identically with means of 1.98 for k-means and 1.99 for the moving method. Therefore, the failure to find a significantly better clustering ability for the moving method in this study is not due to the use of a different measure of clustering performance. Nonetheless, because of its faster convergence, moving methods might be considered a more appropriate benchmark in comparison studies where computational efficiency is a prime factor.

6.5 Conclusions

Although the moving method possesses more convergence states than k-means, at least under the conditions studied here, this translates into faster convergence and not improved cluster recovery as measured by both the Adjusted Rand Index and the within cluster error. Nonetheless, when applying APC to large data sets, the faster convergence is an advantageous property. Because of its faster convergence speed and good clustering ability compared to the other methods examined it is argued that the moving method is the preferable initial partitioning algorithm for use in APC.

In addition to convergence speed, significant effects for cluster shape and sample size were also found. Cluster shape can effect the relative performances of the methods examined. This study only evaluated three shapes and a mixed condition. Other data structures should be investigated in future work. Small data sets can bias performance towards k-means and possibly the moving method as well. This study demonstrated that the better performance of k-means relative to WTANN on the Balakrishnan *et al* (1994) paper was probably due to this factor and not hyperellipsoidal cluster shapes.

An interesting interaction between the neighbourhood function used in KSONN and cluster overlap was also suggested by the results. Increasing the dispersion and Gaussian noise, which increases cluster overlap, improved the performance of this network. However, increasing the number of clusters which also increases the likelihood of cluster overlap did not improve the performance of KSONN. Future work looking at the effects of increasing the learning parameter after the neighbourhood has reached 0 or seeding k-means or the moving method with the centroids found at this point might shed some light of this effect.

It should also be noted that the two neural networks examined are not necessarily representative of all unsupervised neural network architectures available. This chapter is not intended to be a comparative survey of all the major unsupervised neural network architectures available. For example more recent models such as GTM: (Bishop *et al,* 1997); TS-SOM: (Koikkalainen, 1994,1995); Growing Grid: (Fritzke, 1995,1991; ART2: (Carpenter and Grossberg, 1987) and FUZZY ART: (Carpenter *et al*, 1991) are not examined. Rather, the contributions of this chapter to the clustering and unsupervised learning research are as follows: first, a demonstration of how the relative performances of various clustering methods can be dependent on cluster shape. Secondly, to argue that the moving method should be used instead of k-means as a benchmark method as it has been demonstrated empirically that it consistently converges faster. Therefore, future benchmark work involving clustering methods in situations where convergence speed is an important factor (*e.g.* very large data sets) should include the moving method. Finally, as both KSONN and WTANN are often used as benchmarks themselves in comparing both new self-organising neural network methods as well as variations on WTANN and KSONN it is important to understand how KSONN and WTANN perform under a variety of conditions such as cluster shape so that these factors can be taken into account in future benchmarking studies.

7. An Empirical Evaluation of APC and Other Hybrid Clustering Methods

7.1 Introduction

This chapter evaluates the clustering abilities of APC and a number of other two stage hybrid methods that use an initial partitioning of the data as inputs to a hierarchical technique to recover cluster structure. The clustering abilities of APC, WHM and other hybrid strategies are evaluated via two Monte Carlo simulations. The first experiment examines the relative clustering abilities of these methods on the same set of clusters shapes used in the previous chapter. The second experiment examines a different set of cluster shapes testing the abilities of the clustering methods to extract clusters that are positioned close together in the pattern space.

7.2 Hybrid methods

The relative clustering abilities of APC, WHM and the two stage hybrid method proposed by Beale (1969) and Wishart (1978) are evaluated. For all of the clustering techniques, the moving method was used to derive *k* initial cluster centroids as it has been shown to converge faster and to minima as good or better than k-means. Unsupervised neural networks were not considered because of the high CPU burden. Once the centroids of the initial clusters were been found they were then agglomerated using the following hierarchical techniques: Ward's method (WARD), average linkage (ALINK), centroid linkage, (CLINK) and single linkage (SLINK). For all these methods, intercluster distance was measured using Euclidean distance.

ALINK, CLINK and WARD were chosen as they are commonly used and have been shown to be very effective agglomeration techniques (Lorr, 1983; Milligan, 1981a). WHM and SLINK were included because they are capable of finding chaining clusters. WHM was also included because like APC, it uses a density based metric of cluster distance. Regarding APC, both the ADD and UDD distance measures were evaluated.

7.3 Experiment 1

In this experiment the same cluster structures and data generation as in the previous chapter was used. It is hypothesised that ALINK, WARD and CLINK will not perform as well as the others on the "chaining" and mixed conditions as they tend to perform best on compact cluster structures. WHM, APC and SLINK should give the best recoveries of these structures.

In addition to cluster shape, four other factors were incorporated into the Monte Carlo simulation. The number of clusters was varied from 3, 5 and 7. Cluster dispersion was varied from σ = 0.025, 0.05 and 0.075. 5 types of noise were also examined. Level 0 was no noise present in the data. Levels 1 and 2 had σ and 2σ Gaussian noise added to 50% of the patterns. This condition was the same as that recommended by Milligan (1985) and is effectively a simulation of measurement error. Level 3 noise was 25% of the patterns replaced with uniform noise in the range of the data set. Level 4 noise had 50% of the patterns replaced with uniform noise. This type of noise simulates uncorrelated structure in the data as was used following the example of Cowgill (1993). The dimensionality of the data was also varied from 4, 7 and 10 dimensions.

In all the Monte Carlo simulation was a 4 (cluster structure) x 5 (noise) x 3 (dispersion) x 3 (dimension) x 3 (number of clusters) factorial model. The number of data patterns (N) was fixed at 1500 and the number of initial partitions, *k*, was set to $10 \cong N^{0.3}$. The value of *k* around $N^{0.3}$ was suggested by Wong (1982) as a general rule of thumb. For each of the 520 cells, 30 replications were made to try and average out the effects of any unusually poor initial partitions for a total of 15,600 data sets generated.

In order to prevent small and insignificant initial clusters being incorporated into the agglomeration process, any of the initial *k* clusters found to contain less than 2% of the data were discarded. The moving method was then run again, using the existing cluster centroids as starting seeds and resetting *k* to *k - 1*. This process was repeated until all clusters contained at least 2% of the patterns[14]. Therefore, although *k* was set to 10 initially, not all agglomerations were done with 10 initial cluster centroids, however, agglomerations made with less than 10 initial partitions are well in the minority.

7.3.1 Results

Table 7.1 displays the overall mean Adjusted Rand Index (ARI) scores of each of the 7 hybrid clustering algorithms over all conditions. The ARI scores were significantly different at $p < 0.01$ for all hybrid methods except UDD and WARD which represent the best performing algorithms at ARI = .653 and .654 respectively. ADD, ALINK and CLINK produced the next best scores with .648, .643 and .637 respectively. SLINK and WHM performed worse with scores of .621 and .564.

UDD	ADD	WARD	ALINK	CLINK	SLINK	WHM
.653 A	.648	.654 A	.643	.637	.621	.564

Table 7.1: Overall mean ARI scores for each clustering algorithm. '*A*' indicates scores not significantly different at 0.01 level.

14 As in the Wishart (1969b) variation of k-means.

	UDD	ADD	WARD	ALINK	CLINK	WHM	SLINK
DATA SET							
Hyperellipsoid	.717	.716	.727	.719	.715	.637	.702
Cone	.589	.578	.567	.550	.539	.470	.516
Arc	.636	.630	.645	.639	.639	.570	.622
Mixed	.672	.668	.677	.665	.657	.580	.643
CLUSTERS							
3	.682	.681	.703	.687	.681	.585	.656
5	.653	.648	.643	.630	.623	.535	.601 a
7	.625	.617	.615	.612	.609	.573	.605 a
DIMENSION							
4	.642	.634	.642	.629	.622	.544	.597
7	.662 a	.658	.662 a	.651 a	.645 a	.569	.633 a
10	.657 a	.652	.658 a	.650 a	.646 a	.580	.632 a
NOISE TYPE							
No noise	.910	.904	.891 a	.886 a	.884 a	.873 a	.880 a
Gauss σ	.900	.889	.893 a	.887 a	.885 a	.867 a	.876 a
Gauss 2σ	.819	.807	.864	.858	.857	.816	.849
25% Uniform	.439	.438	.434	.409	.400	.203	.353
50% Uniform	.200	.203	.187	.176	.163	.063	.145
DISPERSION							
0.025	.673	.670	.660 a	.650 a	.644 a	.579	.629 a
0.05	.659	.653	.658 a	.648 a	.642 a	.572	.625 a
0.075	.629	.622	.643	.632	.626	.542	.608

Table 7.2: Mean ARI scores for each algorithm on the first test suite taken over each condition. Scores in the same column under the same condition with the same letter are not significantly different at the 0.01 level.

Table 7.2 and Table 7.3 give the mean ARI score of each hybrid algorithm over each condition. An ANOVA (analysis of variance) was run to examine the effect of each level on the various algorithms and to compare their relative performances at each level of each condition. Table 7.2 shows the effects of the level of each condition on each clustering algorithm. Mean ARI scores for the same algorithm under the same condition with the same letter are not significantly different at the 0.01 level. Table 7.3 indicates relative differences in cluster recovery performance between clustering algorithms across each *level* of each condition. Algorithms with mean ARI scores with the same letter in each row are not significantly different. For the cluster shape condition, all of the hybrid clustering algorithms performed similarly in that all performed best on the Gaussian shapes, next best on the mixed condition, next best on the arc and worst on the cones (Table 7.2).

	UDD	**ADD**	**WARD**	**ALINK**	**CLINK**	**WHM**	**SLINK**
DATA SET							
Hyperellipsoid	.717 A	.716 A	.727 A	.719 A	.715 A	.637	.702 A
Cone	.589 A	.578 A	.567 AB	.550 BC	.539 CD	.470	.516 D
Arc	.636 A	.630 A	.645 A	.639 A	.639 A	.570	.622 A
Mixed	.672 A	.668 AB	.677 A	.665 AB	.657 AB	.580	.643 B
CLUSTERS							
3	.682 A	.681 AB	.703 A	.687 AB	.681 A	.585	.656 B
5	.653 A	.648 AB	.643ABC	.630 BC	.623 CD	.535	.601 D
7	.625 A	.617 A	.615 A	.612 A	.609 A	.573	.605 A
DIMENSION							
4	.642 A	.634 A	.642 A	.629 A	.622 A	.544	.597
7	.662 A	.658 A	.662 A	.651 AB	.645 AB	.569	.633 B
10	.657 A	.652 AB	.658 A	.650 AB	.646 AB	.580	.632 B
NOISE TYPE							
No noise	.910 A	.904 A	.891 B	.886 BC	.884 BC	.873 C	.880 BC
Gauss σ	.900 A	.889ABC	.893 AB	.887ABC	.885 BC	.867 D	.876 CD
Gauss 2σ	.819 B	.807 B	.864 A	.858 A	.857 A	.816 B	.849 A
25% Uniform	.439 A	.438 A	.434 A	.409 B	.400 B	.203	.353
50% Uniform	.200 A	.203 A	.187	.176	.163	.063	.145
DISPERSION							
0.025	.673 A	.670 A	.660 AB	.650ABC	.644 BC	.579	.629 C
0.05	.659 A	.653 A	.658 A	.648 AB	.642 AB	.572	.625 B
0.075	.629 AB	.622 AB	.643 A	.632 A	.626 AB	.542	.608 B

Table 7.3: Mean ARI scores for each algorithm on the first test suite tested for significance across the same level of each condition. Scores in the same row with the same letter are not significantly different at the 0.01 level.

Interestingly, WHM did not outperform CLINK, ALINK or WARD on any of the shape conditions including the cones and arcs (Table 7.3) and was the worst performing algorithm for all shape conditions. SLINK generally performed as well as CLINK and ALINK except under the cone shaped clusters. UDD, ADD and WARD were the best performing algorithms under all shape conditions. It is also interesting to point out that under the hyperellipsoids, there were no significant differences in performance between any methods except WHM. However, under other shape conditions, significant differences in performance begin to show up. Again, this highlights the effect cluster shape can have on the abilities of clustering algorithms to recover cluster structure.

The number of clusters also effected each of the algorithms similarly with performance falling with the increase in the number of clusters although for SLINK there was no significant difference between 5 and 7 clusters. This result is similar to that found by Cowgill (1993) and Balakrishnan *et al* (1994) with k-means and self-organising neural networks and Chen *et al* (1995) and Milligan *et al* (1983) with hierarchical methods. Relative performances across clustering algorithms followed similar patterns as before with ADD, UDD and WARD always appearing in the best performing group and WHM always performing the worst.

The effects of dimension across algorithms was also similar. In general, increasing the number of dimensions significantly improved performance. This would be expected as increasing the dimensionality increases the amount of information available about the cluster structures. All methods except WHM and SLINK performed best with no significant differences in cluster recovery.

Under the noise conditions, all methods performed best under no noise. Increasing the level of noise under both types of noise, similarly decreased the cluster recovery rate for all clustering methods.

Increasing the dispersion also reduced the cluster recovery of each algorithm. This effect was fairly constant across algorithms. Note, however, that this decrease in performance was not statistically significant between the 0.025 and 0.05 levels for ALINK, CLINK, WARD and SLINK. Even where statistically significant effects of dispersion were found, the absolute differences in performance means were not large (*i.e.* 2% - 3%). The relative performances across algorithms followed similar patterns as before.

Under the noise conditions UDD and ADD were effected by all changes in level of noise. The other clustering methods were also significantly effected by all changes in noise level except between the no noise and Gauss σ levels. Regarding relative performances across algorithms, UDD and ADD were in the best performing group under all conditions except Gauss 2σ. Here all of the Euclidean based methods performed best while ADD, UDD and WHM all performed at the same level but worse than the Euclidean methods.

7.3.2 Discussion

UDD, ADD, WARD, ALINK, CLINK and arguably SLINK all performed relatively equally well. The difference in overall mean performance between UDD and SLINK is only 3%. The difference in performance between the others (except WHM) is even less so. Although the differences in overall mean ARI score are statistically significant (except WARD and UDD) the lack of clear cut differences in performances of the above methods makes it difficult to make statements regarding the superiority of one method over another. If one looks across Table 7.3 along the rows, ADD, UDD and WARD produce higher mean ARI scores over the other methods fairly consistently. However, these differences are often small and not always statistically significant (the largest differences in performance become apparent under the noise conditions). It is the consistency as opposed to magnitude of performance of the group comprising ADD, UDD and WARD over the group consisting of ALINK, CLINK and SLINK that highlights the first groups possible superiority under these test conditions. Only WHM consistently under performed the other methods by a large magnitude (5% - 10%).

The relatively equal performances of the hybrid methods was probably due to the high degree of cluster separation in the data. A lack of interaction effects involving dispersion and the number of clusters found in an ANOVA model run on the results (not shown) seems to support this. One would expect an interaction between these two factors as increasing the dispersion and the number of clusters would increase the degree of cluster overlap. This in turn would decrease cluster recovery. In any case, the results of experiment 1 suggest that when clusters are well separated, one only needs Euclidean distance information between clusters to make accurate classifications when using the two stage type of hybrid method explored in this work.

Only WHM performed at sufficiently poor levels to allow one to doubt its effectiveness in this type of clustering problem. The mean ARI score of WHM was over 5% below that of SLINK and about 10% worse than UDD or ADD. This suggests that the use of more local information in estimating the intercluster densities can significantly improve the performance of WHM's method. In other words, the relatively poor performance of WHM indicates that estimating the intercluster density at only the midpoint between cluster centroids can be inadequate in some situations. WARD, ALINK, CLINK and SLINK probably performed better than WHM because the high degree of cluster separation meant that intercluster distance was proportional to the intercluster density. Therefore one only needed to look at the distances between the centroids of the initial moving method derived clusters to determine which centroids belonged together. The fact that the Euclidean distance based methods outperformed WHM further highlights the ineffectiveness of the density based metric used by WHM.

Regarding the low cluster recovery on all methods in the uniform noise conditions, it should be taken into account the relatively low scores in the uniform noise conditions reflect the fact that the maximum possible ARI scores would be on the order of .750 and .500 for the 25% and 50% levels. This is due to the patterns being randomly scattered around the pattern space. Scores of about .400 and .200 are similar to those found by Cowgill (1993) whose use of uniform noise was followed here. Nonetheless, this does show that all of the hybrid methods are severely effected by uniform noise. This may be due in large part to the pattern partitioning stage. The presence of uniform noise may be leading to large numbers of cluster centroids being placed in regions between clusters causing all of the methods to chain clusters together.

Finally, the number of initial partitions k was not varied in this experiment. Although Wong (1982) suggested that $k \cong N^{0.3}$ is a good guideline, Wong also demonstrates empirically that in some cases a much larger value is needed to detect finer cluster structures. In fact, the decrease in performance with the increase in the number of clusters may be related to the fixed value of k. Increasing the number of clusters increases the overall complexity of the pattern space the hybrid algorithms must operate in. As the conditions with larger numbers of clusters require the hybrid methods to model a more complex space, a larger value of k is probably needed. In other words, k = 10 may have been too small for the 5 and 7 cluster conditions and therefore led to an over smoothing of the cluster structures.

7.4 Experiment 2

A second experiment was run to try and obtain a better idea of the relative clustering abilities of the hybrid methods. In this experiment, long chaining clusters are placed in close proximity to other clusters. Whereas before the clusters in the Monte Carlo simulation exhibited properties of cohesion and isolation, the cluster structures in this experiment only exhibited cohesion. The clusters were intentionally placed in close proximity (*i.e.* without isolation) to test the ability of the various hybrid techniques to discriminate between closely placed clusters. The relative proximity of the clusters is also effected by the dispersion parameter.

It is hypothesised that by placing chaining clusters close together, CLINK, ALINK and WARD will fail because the distances between the initial pattern partitioning derived cluster centroids will no longer reflect the cluster structure. In addition, it is expected that SLINK will outperform CLINK, ALINK and WARD as the chaining properties of this method should now operate in its favour. WHM should outperform SLINK as SLINK does not take intercluster density into account in the agglomeration process. Also, SLINK may start to chain clusters together in the presence of noise and when cluster dispersion is relatively high. Finally, it is expected that APC (ADD and UDD) will produce better cluster recovery than WHM or SLINK as APC's measure of intercluster density is more accurate than that used in WHM and because it is not as likely to chain clusters together than SLINK.

Unlike experiment 1 the number of initial partitions, *k*, will also be varied. As 1500 patterns will be used as before, *k* will be set to 10, 15 and 20 representing values of 1.0, 1.5 and 2.0 times the recommended value of $N^{0.3}$. Note that *k* operates as a smoothing factor. Smaller values of *k* will have the tendency to smooth out local structures. Too small a value will lead to important cluster structured being ignored.

High values of *k* make the model sensitive to local information although too high a value can result in spurious local structures being overemphasised.

The new data sets to be used are displayed in Figure 7.1. They are (a) concentric clusters, (b) three parallel linearly separable chaining clusters, (c) two interlocking, nonlinearly separable chaining clusters and (d) a winding chaining cluster bordered by two Gaussian clusters. These cluster structures were inspired by an informal comparative study in Backer (1995). Unlike experiment 1 the performance of the clustering methods will be evaluated on each data set individually.

These cluster shapes were constructed similarly as in the previous experiment. The "ring" in the concentric clusters was generated by extending the "arc" cluster shape to 360 degrees. The winding and interlocking clusters were constructed using 240 degree arcs although the winding cluster shape required two arcs to be linked together. The Gaussian clusters were generated identically as before. Appendix A3 contains the code used to generate the data.

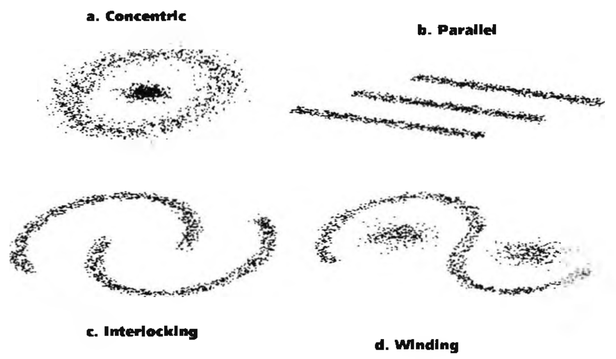


Figure 7.1: Cluster structures used in Experiment 2.

For each of these data sets *k* was varied from 10, 15, 20; dispersion 0.025. 0.05 0.075; dimension 4, 7, 10 and noise type as before. In all a 3 x 3 x 3 x 5 factorial Monte Carlo simulation was run on each data set. As before, each of the 135 cells was repeated 30 times to average out the effects of any poor final convergence states in the initial pattern partitioning for a total of 4050 x 4 data sets in experiment 2.

As with the previous experiment, any initial clusters with less than 2% of the data patterns were discarded. Therefore, the 10, 15, 20 values of *k* only refer to the initial number of run on the data. Generally, the number of clusters that were actually used in the agglomeration stage were equal to or at least proportional to the value of *k* that was begun with.

7.4.1 Results

Table 7.4 gives the mean ARI scores for each algorithm on each data set. On the concentric data set UDD performed best (ARI = .591) followed by ADD (ARI = .539). Given that an ARI score near 0 indicates chance agreement, WARD (ARI = -.034), ALINK (ARI = -.039), CLINK (ARI = -.049) and WHM (ARI = .012) generally failed to model the cluster structures. SLINK (ARI = 0.112) was able to recover slightly more cluster structure than chance. On the Parallel set UDD and ADD perform the best at ARI = 0.673 and 0.670 with no significant difference between them. SLINK (.590) and WHM (.535) showed better cluster recovery than WARD (.504), ALINK (.474) or CLINK (.484). UDD and ADD performed best on the Interlocking data with ARI scores of 0.458 and 0.444. WARD, ALINK and CLINK performed next best with no significant difference in their scores at 0.155, 0.164 and 0.156. SLINK and WHM produced scores of 0.114 and 0.115. Only UDD and ADD were able to perform at significantly above chance levels on the Winding data set with ARI scores of .460 and .427 respectively. WARD, ALINK, CLINK, WHM and SLINK are at or near chance levels. The ARI scores of 0.460 and 0.427

for UDD and ADD show that although these methods could recover some of the cluster structure, they too had great difficulty with this cluster configuration.

SET	UDD	ADD	WARD	ALINK	CLINK	SLINK	WHM
Concentric	.591	.539	-.034 A	-.039 AB	-.049 B	.112	.012
Parallel	.673 A	.670 A	.504	.474	.484	.590	.535
Int. Lock	.458	.444	.155 A	.164 A	.156 A	.114 B	.115 B
Winding	.460	.427	.100	0.068	.048	-.041	-.058
Overall Mean	.552	.520	.181	.167	.160	.194	.151
Overall Mean: Noise = 0	.882	.871	.256	.239	.233	.321	.275

Table 7.4: Mean Adjusted Rand scores of each hybrid method on each data set along with the overall means and the mean performances under no noise condition (see discussion).

Table 7.5 - Table 7.12 give the relative performances of the hybrid methods on each of the 4 data sets individually. Unlike experiment 1, the effects of dispersion, number of partitions, noise and dimensionality on the clustering methods is examined over each of the data set separately.

7.4.1.1 Concentric data

On the concentric data set (Table 7.5 and Table 7.6) WARD, ALINK, CLINK and WHM are performing at or near chance levels under almost all levels of all conditions. WHM and SLINK performed only slightly better. UDD and ADD performed reasonably well under all conditions except under the 25% uniform noise.

Increasing the number of partitions generally decreased the performance of UDD and ADD while the opposite occurred for SLINK. SLINK performed at chance levels at 10 partitions (ARI = -.035) and significantly increased its performance at k = 15 (ARI = 0.120) and 20 (ARI = .252). Increasing the dimensionality from 4 to 10 actually decreased the performance of UDD and ADD by up to 10%. SLINK was only able to extract better than chance levels of structure at dimension = 4 (ARI = .116 and .288).

	UDD	ADD	WARD	ALINK	CLINK	SLINK	WHM
PARTITION							
10	.600 a	.568	-.042	-.042 a	-.054	-.035	-.046
15	.602 a	.541	-.027	-.038 b	-.047 a	.120	-.002
20	.571	.509	-.034	-.038 a b	-.045 a	.252	.083
DIMENSION							
4	.653	.597	-.023	-029	-.038	.288	.116
7	.583	.532	-.039 a	-042	-.053 a	.072	-.034 a
10	.537	.489	-.041 a	-.047	-.054 a	-.023	-.047 a
NOISE TYPE							
No noise	.954	.907	-.025a b	-.031 a	-.036 a	.246	.070
Gauss σ	.900	.800	-.023 a	-.028 a	-.036 a	.206 a	.048
Gauss 2σ	.775	.665	-.028 b	-.032 a	-.038 a	.170 a	-.004
25% Uniform	.092	.086	-.047 c	-.054 b	-.070	-.027 b	-.027 a
50% Uniform	.233	.239	-.047 c	-.051 b	-.062	-.033 b	-.028 a
DISPERSION							
0.025	.675	.676	-.035 a	-.039 a	-.050 a	.165	.060
0.05	.643	.629	-.034 a	-.039 a	-.408 a	.114	.007
0.075	.454	.313	-.033 a	-.039 a	-.049 a	.058	-.031

Table 7.5: Concentric data set results - effects of each experimental factor on each clustering algorithm.

SLINK was able to extract some structure under the no noise and Gaussian noise conditions but failed completely under the uniform noise conditions. Both ADD and UDD performed similarly: the best performance was under the no noise condition while sdding and increasing Gaussian noise decreased performance of both methods. Under the uniform noise conditions, both methods performed better under 50% noise (ARI = .233 and .239 respectively) than they did under 25% noise (.092 and .086 respectively) where they performed at chance levels. Increasing the dispersion decreased the performance of ADD, UDD and SLINK.

	UDD	ADD	WARD	ALINK	CLINK	SLINK	WHM
PARTITION							
10	.600 A	.568 A	-.042	-.042	-.054	-.035	-.046
15	.602	.541	-.027 AB	-.038 AB	-.047 B	.120	-.002 A
20	.571	.509	-.034	-.038 A	-.045 A	.252	.083
DIMENSION							
4	.653	.597	-.023 A	-029 A	-.038 A	.288	.116
7	.583	.532	-.039 A	-042 A	-.053 A	.072	-.034 A
10	.537	.489	-.041 A	-.047A	-.054 A	-.023 A	-.047 A
NOISE TYPE							
No noise	.954	.907	-.025 A	-.031 A	-.036 A	.246	.070
Gauss σ	.900	.800	-.023 A	-.028 A	-.036 A	.206	.048
Gauss 2σ	.775	.665	-.028 A	-.032 A	-.038 A	.170	-.004 A
25% Uniform	.092 A	.086 A	-.047 BC	-.054 C	-.070 C	-.027 B	-.027 B
50% Uniform	.233 A	.239 A	-.047 BC	-.051 BC	-.062 C	-.033 B	-.028 B
DISPERSION							
0.025	.675 A	.676 A	-.035 B	-.039 B	-.050 B	.165	.060
0.05	.643 A	.629 A	-.034 B	-.039 B	-.408 B	.114	.007
0.075	.454	.313	-.033 A	-.039 A	-.049 A	.058	-.031 A

Table 7.6: Concentric data set - relative performance of each clustering algorithm on each level of each experimental factor.

7.4.1.2 Parallel data

Under the parallel data (Table 7.7 and Table 7.8) increasing the number of initial cluster partitions improved the performance of all clustering methods except CLINK and ALINK. Increasing the dimensionality also led to improvements in performance although to varying degrees depending on the clustering method. For ADD and UDD this effect was very marginal. WHM and SLINK improved their performances by 13% and 25% respectively while ALINK, WARD and CLINK all saw improvement of over 100% between 4 and 10 dimensions.

Adding Gaussian noise had similar effects across all methods. Adding 25% Gaussian noise had no significant effects and adding 50% only decreased performance by about 10%. 25% uniform noise led to similar recovery levels for all methods, 50% uniform noise gave all the methods serious difficulties with all methods producing ARI scores at or near chance levels. Increasing the dispersion also caused all of the methods to produce lower ARI scores although none experienced greater than 10% drops in performance between dispersions of 0.025 to 0.075.

	UDD	**ADD**	**WARD**	**ALINK**	**CLINK**	**SLINK**	**WHM**
PARTITION							
10	.660	.657	.493 a	.478 a	.485 a	.509	.548
15	.676 a	.671 a	.505 a b	.465 a	.481 a	.616	.567
20	.683 a	.676 a	.513 b	.478 a	.485 a	.646	.581
DIMENSION							
4	.661	.648	.322	.283	.300	.475	.470
7	.678 a	.676 a	.486	.464	.432	.570	.537
10	.680 a	.680 a	.702	.675	.719	.726	.598
NOISE TYPE							
No noise	.997 a	.997 a	.708 a	.681 a	.681 a	.892 a	.910 a
Gauss σ	.990 a	.981 a	.701 a	.664 a	.675 a	.895 a	.917 a
Gauss 2σ	.922	.901	.657	.614	.644	.846	.834
25% Uniform	.308	.308	.324	.298	.317	.268	.013 b
50% Uniform	.147	.152	.128	.112	.101	.050	.000 b
DISPERSION							
0.025	.690 a	.691	.510 a	.491	.495	.601 a	.553 a
0.05	.682 a	.679	.508 a	.473	.484	.599 a	.547 a
0.075	.648	.634	.492	.457	.472	.571	.504

Table 7.7: Parallel data set - effects of each experimental factor on each clustering algorithm.

The relative performances of the hybrid algorithms was fairly constant with this data. ADD and UDD generally performed best with no significant differences between their performances under any conditions. SLINK was nearly always produced the second best recovery. The same was true of WHM although it did recover significantly less cluster structure than SLINK for 10 partitions and the two highest levels of dispersion. The other methods generally performed worse than SLINK, WHM and APC however their relative performance did seem to improve with increasing the dimensionally of the data.

	UDD	ADD	WARD	ALINK	CLINK	SLINK	WHM
PARTITION							
10	.660 A	.657 A	.493 BC	.478 BC	.485 BC	.509 B	.548 C
15	.676 A	.671 A	.505 C	.465 C	.481 C	.616 B	.567 B
20	.683 A	.676 A	.513 C	.478 C	.485 C	.646 A	.581 B
DIMENSION							
4	.661 A	.648 A	.322 C	.283 C	.300 C	.475 B	.470 B
7	.678 A	.676 A	.486 C	.464 CD	.432 D	.570 B	.537 B
10	.680 A	.680 A	.702 A	.675 A	.719 A	.726 A	.598
NOISE TYPE							
No noise	.997 A	.997 A	.708 C	.681 C	.681 C	.892 B	.910 B
Gauss σ	.990 A	.981 A	.701 C	.664 D	.675 CD	.895 B	.917 B
Gauss 2σ	.922 A	.901 A	.657 C	.614 D	.644 CB	.846 B	.834 B
25% Uniform	.308 AB	.308 AB	.324 A	.298 B	.317 AB	.268	.013
50% Uniform	.147 A	.152 A	.128	.112	.101	.050	.000
DISPERSION							
0.025	.690 A	.691 A	.510 CD	.491 D	.495 D	.601 B	.553 BC
0.05	.682 A	.679 A	.508 BC	.473 C	.484 C	.599	.547 C
0.075	.648 A	.634 A	.492 B	.457 B	.472 B	.571	.504 B

Table 7.8: Parallel data set - relative performance of each clustering algorithm on each level of each experimental factor.

7.4.1.3 Interlocking data

In the interlocking set (Table 7.9 and Table 7.10) all methods other than UDD and ADD generally performed poorly. Increasing the number of partitions significantly improved the performance of all the clustering methods although SLINK and WHM operated near chance for all levels. Interestingly, increasing the dimensionality *decreased* the ARI scores of all the methods, particularly UDD, ADD, SLINK and WHM. In fact, both WHM and SLINK only perform at above chance levels on 4 dimensions.

	UDD	ADD	WARD	ALINK	CLINK	SLINK	WHM
PARTITION							
10	.316	.308	.143	.158 a	.149	.079 a	.079
15	.500	.484	.166	.169 b	.160 a	.092 a	.098
20	.559	.540	.155	.165 a b	.159 a	.170	.067
DIMENSION							
4	.614	.590	.174	.194	.166 a	.225	.239
7	.430	.423	.143 a	.148 a	.138	.065 a	.058 a
10	.331	.319	.147 a	.151 a	.164 a	.051 a	.046 a
NOISE TYPE							
No noise	.813 a	.812	.206 a	.219 a	.205 a	.195 a	.196 a
Gauss σ	.785 a	.763	.200 a	.214 a	.201 a	.185 a	.198 a
Gauss 2σ	.561	.508	.196 a	.212 a	.200 a	.177 a	.178 a
25% Uniform	.106	.107	.119	.123	.122	.003 b	.000 b
50% Uniform	.026	.030	.053	.053	.052	.008 b	.000 b
DISPERSION							
0.025	.525	.523	.155 a	.169 a	.159 a	.124 a	.121 a
0.05	.469	.448	.157 a	.165 a b	.158 a b	.112 a b	.117 a
0.075	.382	.362	.152 a	.158 b	.151 b	.104 b	.105 a

Table 7.9: Interlocking data set - effects of each experimental factor on each clustering algorithm.

The addition of Gaussian nose only effected ADD and UDD although none of the other methods were performing very well at the no noise level anyway. Uniform noise seriously disrupted the performance of all methods with none of the methods performing much better than chance at either the 25% or 50% levels. Increasing the level of dispersion significantly decreased the performance of all of the clustering methods except WARD and WHM. Overall, increasing the dispersion had a greater effect on ADD and UDD although none of the other methods performed well on any of the dispersion levels.

Relative to the other hybrid methods APC (UDD and ADD) outperformed all others under all levels of all conditions except 50% uniform noise. The relative performances of the other methods was less systematic although in general it was more likely for the WARD, ALINK, CLINK group to perform as well or better than SLINK and WHM.

	UDD	ADD	WARD	ALINK	CLINK	SLINK	WHM
PARTITION							
10	.316 A	.308 A	.143 B	.158 B	.149 B	.079 C	.079 C
15	.500 A	.484 A	.166 B	.169 B	.160 B	.092 C	.098 C
20	.559 A	.540 A	.155 B	.165 B	.159 B	.170 B	.067 B
DIMENSION							
4	.614 A	.590 A	.174 D	.194CD	.166 D	.225 BC	.239 B
7	.430 A	.423 A	.143 B	.148 B	.138 B	.065 C	.058 C
10	.331 A	.319 A	.147 B	.151 B	.164 B	.051 C	.046 C
NOISE TYPE							
No noise	.813 A	.812 A	.206 B	.219 B	.205 B	.195 B	.196 AB
Gauss σ	.785 A	.763 A	.200 B	.214 B	.201 B	.185 B	.198 B
Gauss 2σ	.561 A	.508 B	.196 C	.212 C	.200 C	.177 C	.178 C
25% Uniform	.106 A	.107 A	.119 A	.123 A	.122 A	.003 B	.000 B
50% Uniform	.026 B	.030 B	.053 A	.053 A	.052 A	.008 C	.000 D
DISPERSION							
0.025	.525 A	.523 A	.155 B	.169 B	.159 B	.124 C	.121 C
0.05	.469 A	.448 A	.157 B	.165 B	.158 B	.112 C	.117 C
0.075	.382 A	.362 A	.152 B	.158 B	.151 B	.104 C	.105 C

Table 7.10: Interlocking data set - relative performance of each clustering algorithm on each level of each experimental factor.

7.4.1.4 Winding data

Finally, Table 7.11 and Table 7.12 display the performances on the winding set. Again, all methods other than UDD and ADD are performing at or near chance levels. Increasing the number of initial partitions increased the performance of both ADD and UDD. Increasing the dimensionality again greatly decreased ADD and UDD performances. Both UDD and ADD were also significantly effected by both types of noise and completely unable to recover cluster structure in the 50% uniform noise condition. As with all other data sets, increasing the dispersion, decreased the performance of ADD and UDD.

	UDD	ADD	WARD	ALINK	CLINK	SLINK	WHM
PARTITION							
10	.260	.246	.101 a	.071 a	.049 a	-.047	-.057
15	.540	.498	.098 a	.065 a	.054 a	-.061	-.073
20	.580	.538	.101 a	.069 a	.041	-.014	-.044
DIMENSION							
4	.612	.568	.096	.078	.082 a	.018	-.037
7	.420	.389	.102 a	.070	.078 a	-.065 a	-.066 a
10	.350	.325	.103 a	.057	-.016	-.074 a	-.072 a
NOISE TYPE							
No noise	.765	.766	.133 a	.087 a	.071 a	-.049 a	-.077 a
Gauss σ	.697	.650	.130 a	.100	.067 a b	-.060 a	-.085 a
Gauss 2σ	.570	.473	.121	.078 a	.059 b	-.064 a	-.102
25% Uniform	.187	.182	.087	.050	.024 c	-.020 b	-.016 b
50% Uniform	.079	.065	.030	.025	.018 c	-.010 b	-.010 b
DISPERSION							
0.025	.530	.528	.105	.070 a	.051 a	-.037 a	-.058 a b
0.05	.497	.469	.100 a	.069 a	.048 a	-.040 a	-.063 b
0.075	.353	.284	.096 a	.066 a	.045 a	-.045 a	-.054 a

Table 7.11: Winding data set - effects of each experimental factor on each clustering algorithm.

	UDD	ADD	WARD	ALINK	CLINK	SLINK	WHM
PARTITION							
10	.260 A	.246 A	.101	.071 B	.049 B	-.047 C	-.057 C
15	.540	.498	.098	.065 A	.054 A	-.061 B	-.073 B
20	.580	.538	.101	.069 A	.041 A	-.014 B	-.044 B
DIMENSION							
4	.612	.568	.096 A	.078 A	.082 A	.018	-.037
7	.420	.389	.102 B	.070 BC	.078 C	-.065 D	-.066 D
10	.350 A	.325	.103	.057	-.016	-.074 A	-.072 A
NOISE TYPE							
No noise	.765 A	.766 A	.133 B	.087 C	.071 C	-.049 D	-.077 D
Gauss σ	.697	.650	.130 AB	.100 A	.067 B	-.060 C	-.085 C
Gauss 2σ	.570	.473	.121 C	.078 A	.059 A	-.064	-.102
25% Uniform	.187 A	.182 A	.087 B	.050 C	.024 C	-.020 D	-.016 D
50% Uniform	.079 A	.065 A	.030 B	.025 B	.018 B	-.010 C	-.010 C
DISPERSION							
0.025	.530 A	.528 A	.105	.070 D	.051	-.037 D	-.058 D
0.05	.497 A	.469 A	.100 B	.069 BC	.048 C	-.040 D	-.063 D
0.075	.353	.284	.096	.066 A	.045 A	-.045 B	-.054 B

Table 7.12: Winding data set - relative performance of each clustering algorithm on each level of each experimental factor.

7.4.2 Discussion

Table 7.4 gives the overall mean ARI scores for the hybrid clustering methods taken over all 4 data sets in experiment 2. Clearly ADD and UDD produced far superior cluster recovery than any of the other methods which seemed to perform rather poorly. Table 7.4 also gives the mean performance of the hybrid algorithms over the 0 noise level condition. This gives an indication of to what degree the cluster structures irrespective of noise are responsible for the difference in performance. Again ADD and UDD are overwhelmingly outperforming the other methods. The poor performances of ALINK, WARD, CLINK and perhaps SLINK can be explained as being due to the fact that the distances between the initial centroids are no longer indicative of cluster structure under these conditions. Therefore these methods were unable to correctly recover the cluster structures. WHM performed surprisingly poorly. It was expected that it would be able to recover cluster structure significantly better than CLINK, ALINK and WARD. However, the clustering ability of WHM only seem to be marginally better than these methods. Again, this is probably due to the unreliability of basing the intercluster density estimate solely on an estimate of the density at the midpoint between clusters. In fact, SLINK, which is nearly identical to WHM except that no density information is used, recovered cluster structure better than WHM.

Given the results of this experiment and the pervious, it would seem that WHM is a rather poor method of hybridising clustering techniques for large data bases. The relatively superior performances of ADD and UDD indicate that much more local information must be taken into account in situations where chaining clusters lie in close proximity to other clusters. As UDD and ADD perform well even when this is not the case, the versatility of ADD and UDD may well be worth the extra computational costs relative to the other hybrid methods studied here.

Algorithm	UDD				ADD				WARD				ALINK				CLINK				SLINK				WHM			
Data set	*C*	*P*	*I*	*W*	*C*	*P*	*I*	*W*	*C*	*P*	*I*	*W*	*C*	*P*	*I*	*W*	*C*	*P*	*I*	*W*	*C*	*P*	*I*	*W*	*C*	*P*	*I*	*W*
Partitions	-	+	+	+	-	+	+	+	c	+	+	*	c	*	+	c	c	*	+	c	+	+	+	c	c	+	c	c
Dimension	-	+	-	-	-	+	-	-	c	+	-	+	c	+	-	c	c	+	*	c	-	+	-	c	-	+	-	c
Dispersion	-	-	-	-	-	-	-	-	c	-	*	-	c	-	-	c	c	-	-	c	-	-	-	c	c	-	*	c

Table 7.13: Summary of the effects of the number of partitions, dimension and dispersion on each of the hybrid methods over each data set (C = concentric, P = parallel, I = interlocking and W = winding). A "+" indicates increasing the factor significantly increased the ARI score, "-" led to a decrease, "*" indicates no significant effect and "c" indicates performance was at or near chance (ARI < .100) at all levels[15].

Table 7.13 summarises the effects dispersion, number of initial partitions and dimensionality had on each hybrid algorithm on each data set. Both UDD and ADD seem to be very sensitive the number of initial partitions. Increasing *k* significantly improved performance on the Parallel, Interlocking and Winding data sets. However increasing *k* marginally decreased the performance on the Concentric set. In all sets other than the Parallel, increasing the dimensionality adversely effected the their performance of ADD and UDD, particularly on the last two data sets. This conflicts with the results in the first test suite where increasing the dimensionality improved performance. It is an interesting result as one would expect the greater mount of redundant information in higher dimensions to improve, not hinder performance as all of the data sets in experiment 2 were intrinsically two dimensional. In all simulations, increasing the dispersion increased the error rate. This is to be expected as larger dispersion values increase the degree of cluster overlap.

It should also be noted that UDD often recovered more cluster structure than ADD in both experiment 1 and experiment 2 . This may be partly due to the fact that uniform distributions were predominately used to create the various cluster shapes. On the hyperellipsoidal clusters in experiment 1, both methods performed almost equally

[15] On the interlocking data set CLINK was not significantly different between 4 and 10 dimensions while 7 was significantly less.

well. However, on the Concentric data set which was a hyperellipsoidal cluster surrounded by a circular cluster generated with normally distributed patterns, UDD still recovered more structure than ADD by a healthy margin. To a certain extent this was also the case with the winding data set. In any case, the good clustering abilities of APC seems to have more to do with the incorporation of local information in the estimate of the intercluster densities, than how this estimate is converted into a distance metric.

Regarding the other hybrid methods examined, their performances were often so poor it is difficult to assess the impact of the experimental design factors on their ability to recover cluster structure. The only data set they all performed reasonably well on was the Parallel set. Based on this they do seem to benefit from larger numbers of initial partitions and dimensionality as did ADD and UDD on this set. On the other data sets performance is too poor to make any further substantive judgements. Again the reason for this is that WARD, CLINK and ALINK all suffer from the problem of tending to find globular or equally sized clusters. When other types structure are present in the data, these methods will fail. In addition, when clusters lie in close proximity, the Euclidean distance between initial cluster centroids may not adequately reflect the true cluster structure.

7.5 Conclusions

The purpose of this chapter was to compare APC's clustering abilities to that of a number of similarly structured techniques. It was found that when clusters are well separated both APC and the use of Ward's method in the hierarchical agglomeration stage perform equally well. However, when clusters are not well separated APC almost always produced superior results to that of any of the other hybrid techniques examined here.

APC can be viewed as a modification of Wong's hybrid method. The reason for APC's (UDD and ADD) superior clustering ability over WHM is probably due to APC's incorporation of more local information is determining the density based measure of intercluster distance. In WHM an estimate of the density only at the midpoint between clusters is estimated. APC on the other hand makes a more direct estimate of the intercluster density incorporating more local information in the density estimate. Although this requires more computational effort than WHM, the greater overhead required is compensated by the significantly improved performance.

As APC can detect both globular and chaining cluster structures, it is less likely to impose structure on the data than the ALINK, CLINK and WARD hybridisations examined here. In addition, it was shown to outperform both the SLINK hybridisation and WHM that can model chaining structures. APC was also shown to be fairly robust under the Gaussian additive noise corresponding to measurement error. However, the uniform noise (uncorrelated structure) did severely effect its performance.

8. The Use of Linked Line Segments for Cluster Representation and Data Reduction

The purpose of this chapter is to evaluate the use of linked line segments for the representation of cluster structure. As it is argued that this representation enables more accurate assessment of the cluster membership of individual observations than centroids, the use of APC and k-means as data reduction methods are compared. These two approaches along with principal component analysis (for benchmarking purposes) are applied to three simulated and two real world data sets and examined in their ability to provide useful features with lower dimensionality than the original data. The results empirically demonstrate that the linked line segment approach can produce more a salient representation of cluster structure than centroids alone.

8.1 Introduction

Regardless of how one searches for clusters in data, the resulting cluster structures are often modelled as their respective centroids. However, the use of centroids in this manner implies the clusters are largely hyperspherical in shape. If the cluster shape is significantly nonhyperspherical, using the distance between individual observations and the centroid as a metric of an observation's cluster membership can be misleading. However, little work has been done on the representation of cluster structures once they have been extracted from the data set. In other words, once the clusters in the data have been identified one often needs some method to represent or model these structures for use in further analysis[16]. This chapter evaluates the use of the linked line segment based model of cluster structure, which is not biased towards any particular cluster shape. The effectiveness of the linked line segment approach is demonstrated in a data reduction exercise using simulated data.

On common technique is to represent each cluster with its centroid. Based on this one can measure the degree to which a given observation is "typical" of the cluster it belongs to by measuring the observation's distance from the centroid. This can be particularly useful when using cluster analysis as a method of feature extraction or data reduction. For example, one might use cluster analysis to expand the dimensionality of the data to remove nonlinearities or to reduce the number of variables to a more manageable size. To do this, the general procedure is to find *k* clusters or groups in the data from which the transformed data is generated by calculating the distance of each observation from the centroid of each cluster. Let $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, ... \mathbf{x}_N\}$ be a set of *d*-dimensional observations where $\mathbf{x}_n = \{x_1, x_2, ... x_d\}$. After an initial partitioning of $\mathbf{X}$ into *k*, mutually exclusive clusters $G = \{g_1, g_2, ... , g_k\}$ the set $\mathbf{Y} = \{\mathbf{y}_1, \mathbf{y}_2, ... \mathbf{y}_N\}$of *k*-dimensional transformed observations is generated such that $\mathbf{y}_N = \{y_1\, y_2, ... y_k\}$,

$$y_k = \left\| \mathbf{x}_N - \mathbf{c}_k \right\| \qquad 8.1$$

where c_k is the centroid of cluster g_k,

$$\mathbf{c}_k = \frac{1}{n_k} \sum_{j=0, j \in g_k}^{n_k} \mathbf{x}_j \qquad 8.2$$

[16]The term "cluster representation" here is meant to refer to the representation of the cluster used to measure the relative cluster memberships of individual patterns. It is not referring the visualisation of cluster structure in two or three dimensions. See Everitt (1993) for a review of data visualisation methods that could be applied to visualising multivariate cluster structures.

and || || is the Euclidean distance.

The use of the centroid in this manner, however, can lead to problems if the cluster is not roughly hyperspherical or "globular" in shape. Using the centroid implies that observations equally distant from the centroid are equally typical. If one's operating definition of a cluster is a region of relatively high density and the cluster is not hypersherical, the distance from the centroid may not adequately measure cluster membership as the value of equation (8.1) does not necessarily reflect the underlying density of the cluster under these conditions.

Another approach would be to measure the membership of an observation by examining its local density. This can be done by defining a hyperspherical region of radius *r* around each observation and counting the number of observations that fall within this region. The higher the number of observations in the region the more typical the given observation is of the cluster. Other density methods based on the distance of an observation's k-nearest neighbours can also be used. For very large data sets, however, density based methods such as these can be computationally expensive.

This chapter evaluates the use of linked line segments to represent cluster structures. The basic idea is to initially cluster the data into a relatively large number of clusters. Next, these initial clusters are agglomerated into larger clusters when they coexist in areas of relatively high density. The newly merged clusters are then modelled as line segments linking up the centroids of the initial smaller clusters. In this way nonhyperspherical cluster shapes can be modelled and the membership of an observation to a given cluster can be calculated without excessive computational cost.

The remainder of this chapter is divided into four sections. Section 8.2 argues for the use of linked line segments when the cluster structures are nonhyperspherical in shape. Section 8.3 outlines using APC and the linked line segment representation as a data reduction technique. Next, in Sections 8.4 and 8.5 the use of the linked line segment cluster representation as a feature extraction / data reduction method is demonstrated on three simulated data sets. Finally, section 8.6 discusses some shortcomings of this method as well some other concluding remarks.

8.2 Cluster Representation with Line Segments

Regardless of what clustering method is utilised, the resulting clusters are often ultimately modelled as being the centroid of the observations comprising the clusters. However, using the centroid of the cluster as a basis for deriving additional information about a cluster or its members can be problematic as using the centroid implies that the cluster is roughly hyperspherical in shape. Assuming one is interested in clusters defined as *natural* clusters or areas of relatively high density (see Everitt, 1993; Hartigan, 1975) the distance of an observation from the centroid should correspond to the underlying density. If the cluster is roughly hyperspherical in shape this will indeed be true. However, if a given cluster is elongated, or has a long, chaining structure this is no longer the case. Subsequent analysis of how individual observations relate to their given cluster may be distorted. To avoid this problem one could use a splitting technique to find a number of subclusters within the cluster whose centroids span the high density area. However, in this case one has now overestimated the number of clusters in the data.

Figure 8.1*a* and *b* display the k-means clustering of a two cluster problem with $k = 5$ and $k = 2$. One cluster is roughly spherical or compact in shape, while the other has an elongated structure. While the $k = 5$ solution has adequately found the areas of high density, there is nothing in the 4 cluster centroids to indicate that the lower 4 clusters are in fact a single cluster. The number of clusters in the data have been overestimated. The $k = 2$ solution appropriately treats the lower cluster as a single cluster, but subsequent analysis will give misleading information regarding the relationship between individual observations and the cluster. This is illustrated in Figure 8.2. Given observations *a* and *b*, the analyst wants to know which observation is more typical of the cluster. If they were to use the distance of the observations to the cluster centroid to make this decision, observation *b* would be judged as being more typical than observation *a* as it is closer to the centroid. This is a misleading result as observation *a* is situated in a much more dense portion of the cluster - assuming that the operating definition of a cluster is a region of relative high density. The $k = 5$ solution in Figure 8.1*b* would have correctly differentiated between the relative memberships of the two observations, however, in this case the number of clusters in the data set have been overestimated.

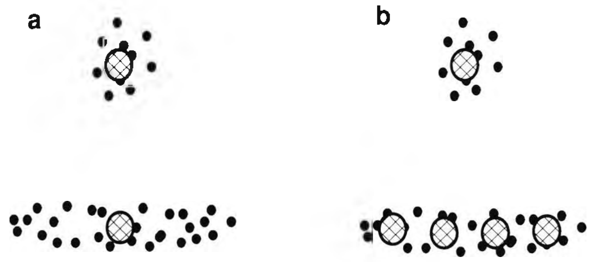


Figure 8.1. Centroids can be inadequate representations of cluster structure.

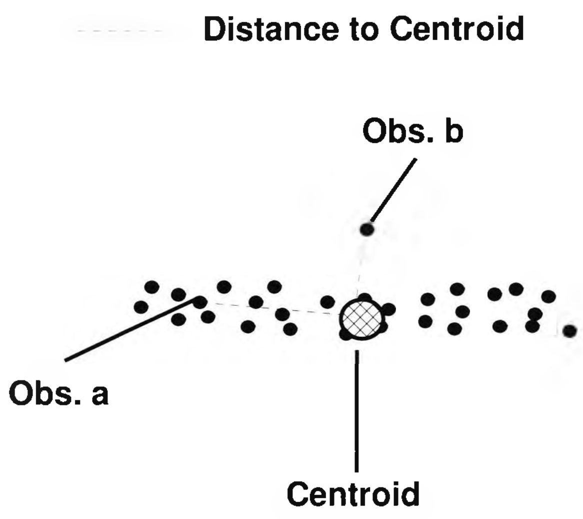


Figure 8.2: Using centroids to measure cluster membership can be misleading.

This trade-off between over estimation of the number of clusters and adequate modelling of cluster structure is a direct result of modelling clusters as centroids. Unless all the clusters in the data set are roughly hyperspherical is shape this dilemma will exist, regardless of what clustering method was used to find the cluster structures. There is a need, therefore, to re-represent the clusters with some other structure that can accommodate nonhyperspherical cluster shapes. The linked line segment representation proposed in this thesis attempts to solve this problem by joining together the centroids of previously agglomerated clusters with line segments when they coexist in contiguous areas of relatively high density. This not only allows the modelling any cluster structure that is piece-wise linear approximatable, but also allows more accurate modelling how individual observations fit into the overall cluster structure of the data.

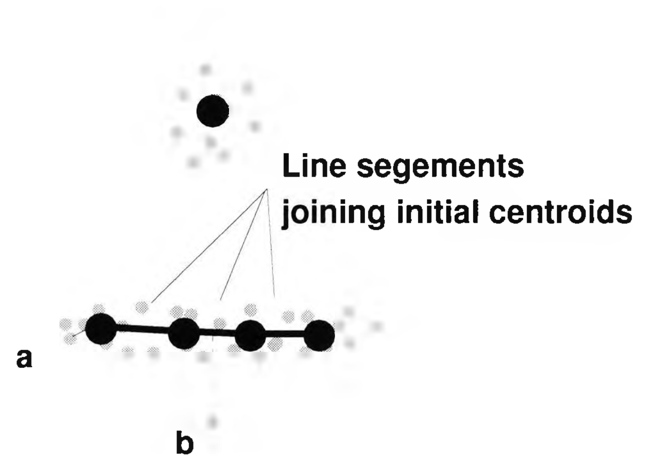


Figure 8.3: Using the linked line segment representation, an outlying pattern's membership is correctly measured without over specifying the number of clusters in the data.

For example, the linked line segment representation of the above two clustering problem is displayed in Figure 8.3. The original k = 5 solution has been agglomerated with line segments joining the original centroids. The elongated cluster is now correctly modelled as being a continuous area of high density. Moreover, if the distance of an individual observation from the cluster is taken as being the distance between it and the nearest line segment in the cluster, the relative memberships of observations *a* and *b* are correctly specified.

8.3 Linked Line Segment Representation as a Data Reduction Technique

In addition to extracting groupings in unknown data sets, cluster analysis can also be used a data reduction or feature extraction method as discussed above. By setting k to less than the number of variables in the data set, one could reduce the dimensionality of the data set to a more manageable size. Alternatively, the goal may be to transform the data into a space that makes the problem easier to solve. Radial Basis Networks, for example, take this later approach (*e.g.* Moody and Darken, 1989).

If one is interested in this approach to feature extraction or data reduction, the problem again arises of how to measure a given pattern's membership or distance from each cluster. As argued above, if the clusters in the data are nonhyperspherical, using the distance of each observation from each cluster's centroid to create the transformed data can be misleading. The use of linked line segments may help alleviate this problem by giving a more faithful representation of the cluster structure to base the transformation on.

8.4 Methods

Example applications of the use of the linked line segments as a data reduction method is given below on three simulated and two real world data sets. The primary purpose of this is to empirically demonstrate that the linked line segment representation of cluster structure can be useful in modelling nonhyperspherical cluster structure. The simulated data sets consisted of intrinsically two dimensional nonlinearly separable clusters embedded in 10 dimensional space. The data sets were the winding, concentric and interlocking clusters used in the previous chapter (Figure 7.1). Three feature extraction methods were applied to extract from 1 to 9 features to generate 9 reduced versions of each data set. These feature sets were then presented to a linear discriminant analysis model (LDA) to test its ability to discriminate between the clusters using the transformed variables. Three feature extraction methods were applied: APC (*i.e.* linked line segments), k-means and Principal Component Analysis (PCA). For k-means, the original data was reduced by initially running k-means with *k* set to the number of desired transformed variables. Once converged, the transformed data was created by taking the distance of each pattern from each cluster centroid. In the APC method, for all number of features, k-means was run with *k* initially set to 20. The resulting 20 initial clusters were then agglomerated using the APC methodology described above. Agglomeration was continued until the desired number of features was found. The transformed variables were then created by finding the distance between each observation to the closest line segment in each cluster.

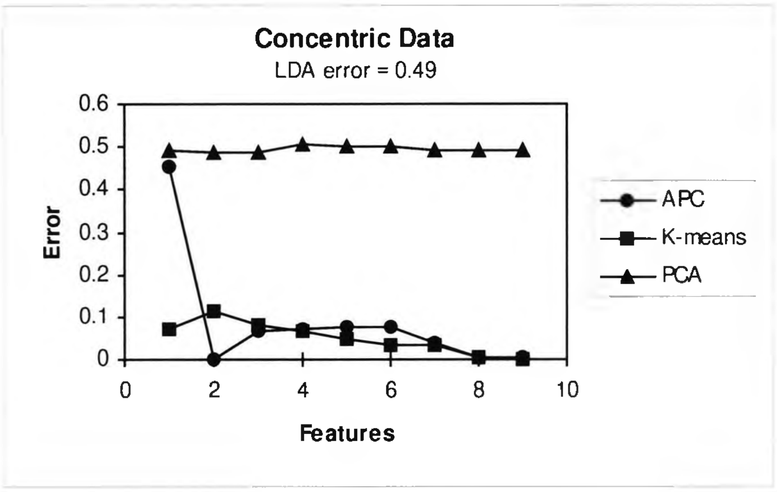
The three data reduction approaches where also applied to the "ionosphere" and "Landsat satellite" data sets available via the UCI Machine Learning Repository (Merz and Murphy (1998)). The Ionosphere and Satellite data consisted of 351 patterns with 34 variables and 6435 patterns and 36 variables respectively. The three data reduction methods were applied to generate reduced versions of both data sets with 30, 25, 20, 15, 10, and 5 features. Regarding APC, *k* was set to 34 for the ionosphere data and 60 for the satellite data.

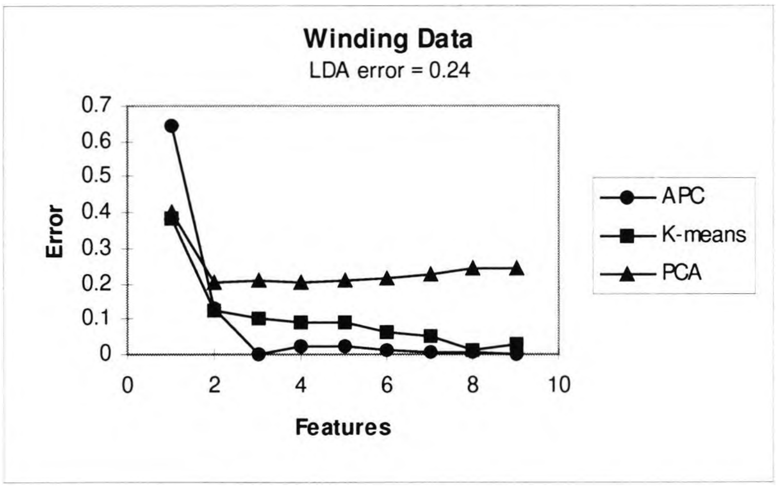
8.5 Results and Discussion

The results of using the reduced simulated and real data sets as input to a LDA model are given in Figure 8.4 and Figure 8.5. Figure 8.4 displays the performance of the LDA model using the 1-9 features derived from each of the data reduction methods on the simulated data. In addition, the performance of LDA on the 10 original variables is also given as a benchmark below each chart title.

On the winding and interlocking data the APC features were able to capture much more of the data structure than k-means. Again, this is due to the linked line segments providing a better "fit" to the cluster structures than is possible with centroids alone. On the concentric data set up until 3 features both k-means and APC perform relatively equally well. At two features APC has captured both the circle and the Gaussian cluster sufficiently well to enable LDA to discriminate between them. The k-means performance at this point is quite poor. However, at one feature the k-means feature leads to only a 7% error rate as opposed to 45% for APC. This is because the circular cluster and the circle can be discriminated with only a single feature as one only needs the distance of each observation from the centroid of the Gaussian cluster. This highlights a limitation of the linked line segment approach in that not all problems are necessarily best modelled is this way. Nonetheless, APC was able to produce 100% accuracy at two features.

PCA generally failed to produce much better results than LDA running on the original 10 variables. This is not surprising, as the data is highly nonlinear. Nonetheless, the PCA derived features did provide better performance than the k-means features on the interlocking data. PCA did not outperform APC on any of the data sets except in the case of only a single feature being used (in the winding and interlocking data). At this point however, both the PCA and APC reduced versions of the data set was no longer outperforming the original 10 variables.





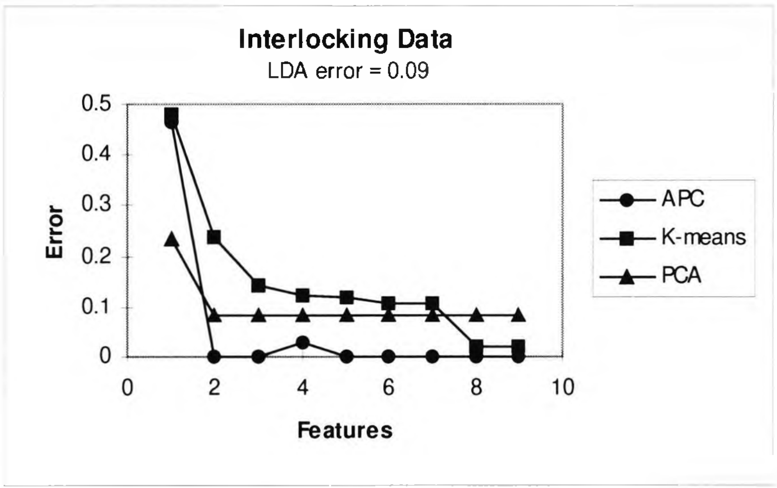
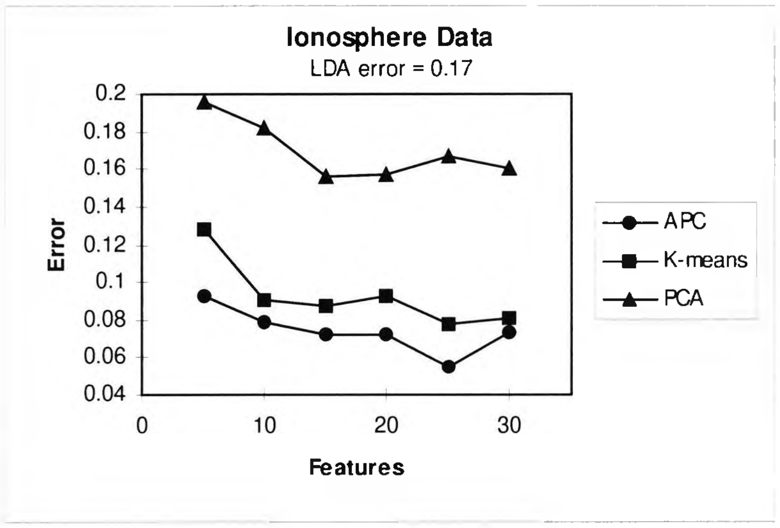


Figure 8.4 The relative performances of the three feature extraction methods on simulated data. The performance of LDA on the 10 original variables is given below the chart titles.



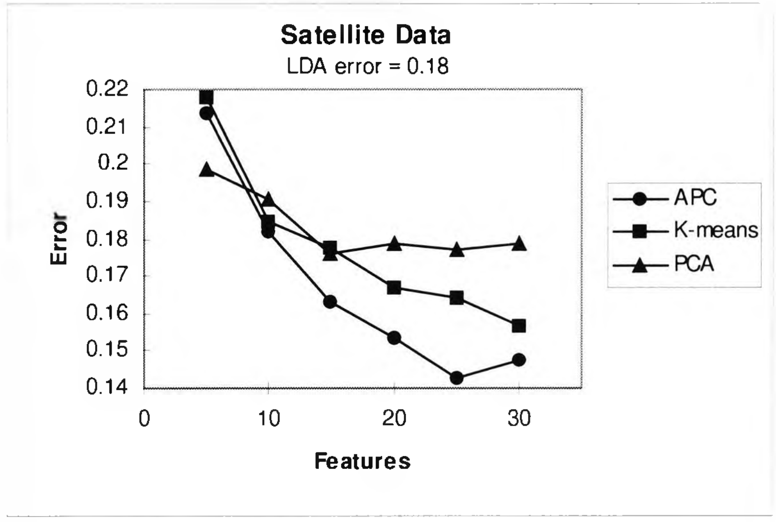


Figure 8.5 The relative performances of the three feature extraction methods on the ionosphere and satellite data.

Figure 8.5 shows the same three methods applied to the ionosphere and satellite data. Both APC and k-means were able to outperform the original 34 variables input to LDA at all number of features tested on the ionosphere data. Similarly, the APC derived features consistently produced lower error scores than k-means and PCA. On the satellite data, the k-means derived features produced lower error scores than the original 36 variables at 30, 25 and 20 features. APC on the other hand produced lower error scores than the original 36 variables and k-means at 30, 25, 20 and 15 features. PCA produced higher error scores than k-means and APC at all feature numbers except 5. However, again at this point all three data reduction methods were producing more errors than the original 36 variables.

8.6 Conclusions

The purpose of this chapter has been to propose the use of linked line segments as an alternative to centroids in the representation of cluster structures. It was argued that when the cluster structure is nonhyperspherical, the use of centroids to measure the cluster membership of individual observations can be misleading. This was empirically demonstrated in the context of a data reduction problem on several simulated data sets where the cluster structures were significantly nonhyperspherical. On all three simulated data sets the linked line segment approach produced superior transformed variables enabling all three nonlinear data sets to be correctly classified by a linear model with only two or three variables. Even on the concentric data set where the centroid based k-means approach was able to model the data with a single feature, it still produced an error rate of 7%, significantly higher than the two variable error rate of 0% by APC. The use of the linked line segment representation was also evaluated on two high dimensional real world data sets. Again, it was shown that the linked line segment representation of cluster structure can be useful in a data reduction context.

A shortcoming of the linked line segment approach however, is that for problems where clusters are well separated and roughly globular in shape, APC will model the clusters with a degree of unnecessary complexity. Also, APC assumes that a problem can broken up into features that correspond to relatively high density clusters. Testing for these conditions, however, can be difficult and costly. Nonetheless, APC is sufficiently efficient that its use as a clustering method or application via the above data reduction paradigm as a data exploratory technique can be justified, particularly on large data samples. For example, as in the case of the ionosphere and satellite data one can compare the relative performances of APC, k-means and PCA in a data reduction context and obtain a general indication of the degree to which the shape of the clusters in the data need be taken account of.

In summary, APC is primarily an unsupervised clustering methodology. The above simulations are meant to highlight the difficulty in modelling nonlinear cluster structures and to show that the use of linked line segments can be useful in this context. In this sense APC should be viewed as a data exploratory method. However, as it can model nonlinear cluster structure it can be useful as a data reduction technique as well. Compared to a number of other approaches to data reduction, APC does have some advantages. For example, APC is computationally efficient - requiring only fractionally more CPU cost than k-means. A number of other methods often used in data reduction contexts such as unsupervised neural networks (*e.g.* Kohonen (1995, 1982), Rumelhart and Zipser (1986)) and nonlinear PCA (*e.g.* Dong and McAvoy (1996)) require many costly iterations through the data. Moreover, because APC can model nonlinear structure is has advantages over linear methods such as PCA and factor analysis as well as centroid based methods such as k-means and unsupervised neural networks. In short, the modelling of nonhyperspherical cluster structures can be problematic. APC with the linked line segment representation of cluster structure can provide an efficient solution to this problem for use in both data clustering and data reduction.

9. Thesis Conclusions and Summary

9.1 Conclusions

As argued in Chapter 3, there is a dearth of clustering algorithms that can simultaneously be applied to large data samples, perform well under noisy conditions and model clusters of arbitrary shape. APC has been proposed as a solution to this problem. As its computational costs are relatively low, it can be feasibly applied to large data samples. By initially clustering the data via fast pattern partitioning methods and then agglomerating these clusters using an efficient method for estimating the intercluster densities, APC has been shown to be an effective data exploratory method for use in situations where most other clustering techniques would be overwhelmed by the computational load. Under a variety of conditions such as noise and nonhyperellipsoidal cluster shapes, the ADD and UDD measures of intercluster density generally lead to more accurate cluster recovery than any of the other hybrid clustering methods examined. This holds particularly true for WHM which also agglomerates clusters using intercluster density. The reason for APC's superior cluster recovery performance relative to WHM is due to the way APC incorporates more local information when estimating the intercluster densities. However, although the relative ability of APC to handle noise in data was to a degree superior to other hybrid methods, under the uniform noise conditions, its performance was disappointingly poor compared to that under Gaussian noise.

This thesis also introduced the concept of using linked line segments to represent cluster structure. This representational scheme allows the representation of any cluster shape that is piecewise linearly approximatable. Chapter 8 demonstrated that on a number of feature extraction problems involving nonlinear data features corresponding to high density clusters, APC with the linked line segment representation was better able to extract these features with fewer parameters than cluster centroids or PCA.

The comparative studies in Chapters 6 and 7 also highlighted the importance of clustering algorithms being able to recover cluster structure as flexibly as possible. Chapters 6 and 7 demonstrated that cluster shape can significantly effect the relative clustering abilities of the clustering techniques examined. The results of the simulations in Chapter 6 also indicated that the moving method consistently converges faster than k-means, suggesting that it may represent not only a more suitable initial pattern partitioning algorithm than k-means for use in APC, but also a more appropriate benchmark for use in evaluating the computational cost of pattern partitioning algorithms in general.

9.2 Limitations

APC was only tested under very high noise conditions and although it generally performed as well or better than other methods, it often failed under the uniform noise conditions which simulate the presence of uncorrelated structure in the data. It could be that the levels of noise were too high and therefore presented too difficult a problem for any clustering algorithm. The use of a more mild level of uniform noise might have given a better picture of how well APC can perform under these conditions. Under the

Gaussian noise, however, APC's performance was much better both in absolute terms and relative to other hybrid methods.

The fact that the empirical comparisons were restricted to other hybrid methods can also be seen as a limiting factor. The reason for this was largely economical - Monte Carlo simulation is a computationally expensive process. Large numbers of simulations had to be done to account for the stochastic nature of the moving method and k-means. Hybrid methods were given priority in the comparative simulations evaluating APC as these are the methods most closely related to APC and because they are the type of clustering approach APC represents an improvement upon. The hybrid methods were also favoured because, at least to the best knowledge of the author, they have never before been empirically compared in a published paper.

Another limitation of this thesis is that alternative approaches to density estimation that could be used in APC instead of histograms (see Appendix A2) were not empirically evaluated. Again, this was because of the computational costs of evaluating additional variations of APC. Additional experimentation could also have been done with a wider range of data and cluster shapes. For example, other types of cluster shape could have been used to gain further understanding of the relative effects cluster shape has on different clustering algorithms. However, as there are an infinite number of possible cluster shapes, any additional experimentation would necessarily be limited by this factor.

9.3 Related Work

The initial stages of APC are similar to Wong's hybrid method. WHM agglomerates an initial set of clusters found via k-means. These initial clusters are then hierarchically clustered via a density based distance metric that is proportional to the density of the observations at the midpoint between clusters. Cluster agglomeration is accomplished using the single linkage method except with agglomerations restricted to neighbouring clusters (see Chapter 3).

APC differs from WHM in how intercluster distance is calculated and in the way clusters are agglomerated. The intercluster distance between two clusters, C_i and C_j, $d^*(C_i, C_j)$, in APC employs a histogram based estimation of the density of observations falling within a hypercylindrical region extending between the centroids of clusters found in the initial pattern partitioning stage. Once the density has been estimated at a number of discrete intervals, $G = \{g_1, g_2, ..., g_m\}$, within this hypercylindrical region, intercluster distance is calculated based on the highest and/or lowest densities found within each of the subcylindrical regions. WHM on the other hand calculates $d^*(C_i, C_j)$ as being proportional to density at the midpoint between the two cluster centroids. APC and WHM also differ in how the initial clusters are agglomerated. WHM uses the single linkage algorithm while APC generates a connected graph. The connected graph approach has the advantage of allowing agglomerations to occur between initial clusters that exists within the same previously agglomerated cluster but have not yet been linked together with a line segment. This enables APC to more faithfully model noncompact cluster shapes.

The two stage process of generating an initial clustering via a pattern partitioning methods followed by the systematic agglomeration of neighbouring clusters based on their centroids has also been previously proposed. Murtagh (1995) has also suggested a hybrid clustering approach where neighbouring outputs of a Kohonen network are hierarchically agglomerated based on their proximity in the output layer (feature map). However, Murtagh's technique is more similar to other approaches of using standard hierarchical techniques to cluster pre-pattern partitioned data (via k-means) as developed by Wishart (1978) and Beale (1969). APC differs from these methods by using a density based metric of intercluster distance as in WHM and in the way APC agglomerates via a connected graph. APC also differs in this respect from Chaudhuri and Chaudhuri (1997, 1995) in that they use a MST based approach to cluster agglomeration. Moreover, their measure of intercluster distance is based on measuring cluster overlap as opposed to density.

The direct searching of discrete regions of the pattern space between clusters in APC to determine the intercluster density is similar in spirit to that of Chaudhuri *et al* (1992) and Liu and Tsai (1989). The difference here being that Chaudhuri *et al* search a number of strips in a finite number of directions from the cluster centroid. Liu and Tsai systematically search linear projections of the data onto orthogonal axis for localised areas of low pattern density. APC on the other hand restricts the search to the spaces between other cluster centroids. Moreover, neither Chaudhuri *et al* (1992) nor Liu and Tsai (1989) use their methods to generate hierarchical agglomerations.

9.4 Future Work

Additional research to extend the findings of this thesis follow from the limitations noted above. Regarding APC, a more detailed examination of the merits of a wide range of intercluster density estimation methods needs to be examined. For example, it would be interesting to test the relative clustering abilities of different versions of APC using nearest neighbour, naive estimates and kernel based methods. Although these method are computationally expensive, it is possible they could provide better performance.

As noted above, APC could to be examined under a wider range of experimental conditions. This could include the use of categorical, binary and mixed data. However, the use of these data types may require modification of the interpattern distance measure and density estimation approach taken here. It would also be particularly interesting to apply APC to specific problem areas where complex data shapes are known to exist. For example, image analysis data often contains highly irregular cluster shapes. As this type of data is often noisy and quite voluminous, APC could provide an effective technique for segmenting image data.

Different implementations of the linked line segment representation could also be explored. APC represents cluster structure with linear approximation of the cluster shape. Other line fitting methods could also be used. For example, once the piecewise linear approximation of the cluster shape has been established, a nonlinear line fitting method could be incorporated to try and get a better fit of the portion of the data identified as a cluster.

Finally, the determination of how one decides on when to stop the agglomeration process was only briefly touched on. Part of the reason for this is that no particular cut-off method particular to APC was proposed in this thesis. However, an evaluation of a wide variety of stopping rules could be examined for use in APC. In fact, what systematic evaluation of stopping rules that have been made (*e.g.* Atlas and Overall, 1994; Milligan and Cooper, 1985; Milligan, 1981b; Mojena, 1977) have not looked at the effects of cluster shape. Given that many of the stopping rules that have been proposed assume multivariate normal clusters, it would be interesting to see how cluster shape interacts with these methods both in APC and in other clustering methods.

A1: Seeding methods for k-means and the moving method

Both k-means and the moving method require an initial partition of the data (referred to as a set of seed points or vectors) to initialise the cluster formation process. As neither method is guaranteed to converge to an optimal minima, it is generally recommended that these methods be run repeatedly with different seed values in order to increase the likelihood of obtaining good convergence (Backer, 1995). The choice of initial seeds is therefore important as it its the initial seed values can determine the quality of convergence state k-means or the moving methods will settle to.

The following is a brief survey of techniques for seeding these methods. Seeding techniques can be divided into two categories based on the underlying strategy of how the seeding should be done (Anderberg, 1973). One approach is to try and ensure "indifference" in regards to how the initial seed points are selected (Doyle, 1966). The idea here is to select seeds in such a way that will not introduce any bias towards one clustering solution or another in the initial configuration. Indifference in this respect can be attempted by either using some degree of random selection in choosing initial seeds or by trying to ensure that the initial seeds are well separated and span the data set. The other approach to finding initial seeds involves some "intelligent" guessing in trying to find initial seed values. These methods generally involve partially solving the clustering problem and then choosing seeds that are likely to fall near where the centroids of the optimal clusters lie.

A1.1 Indifference methods

Indifference methods generally involve choosing patterns from the data set to serve as seed points. Various ways of doing this have been proposed such as choosing the first *k* patterns (MacQueen, 1967), every *(n/k)th* pattern (Anderberg, 1973) and randomly selected patterns (McRae, 1971). Random pattern selection has the advantage of being quick, simple and ideally suited for running multiple clusterings to help overcome convergence to local minima.

Anderberg (1973) also discusses number of other indifference methods for seeding k-means which involve finding a set of seed patterns that span the data set. Anderberg suggests accomplishing this by synthetically creating a set of seed points that span the data set. Another method is to randomly choose patterns from the data set but only using those seeds that are well separated from each other. To do this Astrahan (1970) suggests first calculating the local density around each pattern. Use the highest density point as the initial seed. Find the next highest density point and use it as the next seed if its distance from the all other seeds is greater than some threshold, *d*. Continue this process until one has the desired number of initial seeds. Ball and Hall (1967) propose a similar method except the initial seed is chosen as the mean vector of the data set. Subsequent seeds are chosen from the remaining data patterns that are of at least *d* distance away from all other seeds.

A1.2 Intelligent Guessing Methods

A number of schemes have been proposed involving the use of an initial clustering method to generate a crude partition of the data whose centroids can then be used as seed values. Lance and Williams (1967b) and Wolfe (1970) explored using hierarchical clustering methods to find the initial clusters. Running on a small random sample of the data (*i.e.* n < 300) may make this approach feasible. Tou and Gonzalez (1974) recommend that the sampling be done with the constraint that sample patterns are at least *d* distant form each other. *d* in this case generally does not have to be as large as that used when finding the seed points directly as with Ball and Hall (1967). Shattuck *et al* (1991) suggest that the choice of *d* when sampling is not crucial when creating an initial sample set.

Evolutionary computing strategies have also been applied to seed selection. Babu and Murty (1993) used a genetic algorithm (GA) based method. Effectively, this approach is equivalent to running multiple runs of k-means via the GA to find improved seed values. The initial seed values are found by dividing up the patterns space into a multidimensional hyperrectangular grid. Seed values are taken from the grid vertices. The GA essentially provides a guided random search of different possible seed values which attempts to ensure good seeds are preserved while bad seeds are discarded.

A1.3 Comparison studies

A number of studies have examined the effects of different seeding methods. Most of these have compared the random pattern approach to the use of hierarchical methods to find initial seeds. Blashfield (1977) found better clustering with random pattern seeding than with using Ward's method to find the initial seeds. Milligan (1980) however found that using average linkage to find initial seeds outperformed the random pattern approach. Results in Scheibler and Schnieder (1985) also indicated that seeding with clusters found by Ward's method performed better than random patterns. Cowgill (1993) also compared using Ward's methods to random seeding. Cowgill found that when clusters were well separated, Ward based seeding performed best. However, when the clusters were of different sizes or allowed to overlap, results were generally mixed. In the case of small numbers of well separated clusters, the likelihood of choosing random seeds that exclude one of the clusters is relatively high. In this case, Ward's method may be a better approach. However, as Ward's method is biased towards equally sized, hyperellipsoidal clusters, under other conditions it may not perform as well.

Finally, Shattuck *et al* (1991) examined seven seeding methods on three data sets involving the categorisation of clay minerals and aerosol particles. The first method called the "merge" method was similar to the Ball and Hall (1967) and Astrahan (1970) methods described above. Here, the two closest patterns were found with the second of the pair rejected as a possible seed. The process is repeated with the remaining patterns and the rest of the data set until the desired number of seeds is left in the pool of possible seeds. The second method examined was the "refine" approach found on the FASTCLUST procedure in the SAS statistics package. The refine method is similar to selecting random patterns as seeds except that temporary clusters are generated by assigning each pattern to its closest, randomly selected seed. The centroids of these clusters are then used as the seeds of the k-means procedure. The other five methods were the use of centroid linkage, single linkage, complete linkage, average linkage and Ward's method run on a sample of the data to find an initial clustering whose centroids were used as initial seeds. Shattuck *et al* concluded that the refine, merge and single linkage seeding methods were best for finding fine cluster structure. The centroid linkage, Ward's and average linkage were best for finding the overall course structure of the data. Ultimately, they argue that it is best to apply both groups of seeding methods in order to ensure a good, comprehensive view of the data, as at least in their application, no single seeding method seemed to perform optimally.

A1.4 Conclusions

Given that many of the comparison studies other than Shattuck *et al* relied on MVN simulated data, it is not surprising that Ward's method of seeding often performed best in previous studies. This is further illustrated by the results of Cowgill's study where Ward's approach performed best on well separated MVN clusters while not as well on overlapping or variably sized clusters. In short, Shattuck *et al*'s conclusions seem to be the most appropriate. There probably is no overall superior seeding method. What works best and when is essentially problem dependant. However, using a number of methods, particularly alternating between one biased towards course structure with one biased towards fine grained structures could help reduce the probability that potentially useful cluster structures are overlooked.

A2: Alternative density estimation methods for use in APC

This appendix gives a brief discussion of a number of commonly used density estimation techniques and examines how they can be incorporated into the intercluster density estimation stage of APC. The various advantages and disadvantages of utilising these different techniques in the context of APC is also explored. However, as the histogram approach is the most efficient computationally and the easiest to implement, it is the only density estimation method used in the empirical sections of this thesis.

A2.1 Density Estimation

Density estimation is a branch of statistics that attempts to estimate the density function of observed data. Density estimation techniques can be divided into two types: parametric and nonparametric. Parametric density estimation begins with the assumption that the distribution of the data belongs to some known family of probability distributions. If one's assumptions are correct, one only then needs to estimate the parameters of that distribution to fit the model to the data. For example, if the data is known or assumed to be normally distributed, the density estimate of the data can be calculated by simply estimating the mean and variance of the data.

Nonparametric density estimation does not assume the data are distributed under some particular family of distributions. Rather, they are bottom up techniques that allow the data to determine the model as opposed to top down parametric techniques that attempt to fit a model to the data. Obviously if one had sound reason to believe that the data between the centroids is distributed in some particular fashion along the lines of a known family of distributions than the use of parametric techniques would be appropriate in APC. However, cluster analysis is primarily used in data exploration where little is known about the data. Therefore, in most situations nonparametric techniques are appropriate and will be only ones discussed here. The range of nonparametric density estimation techniques is far to large to give a complete survey. This discussion will concentrate on naive estimators, kernel estimators and nearest neighbour methods. A discussion of the use of histograms has already been given in Chapter 4. For a survey of the other methods as well as a more thorough treatment of density estimation in general, see Silverman (1986).

A2.2 The Naive Estimator

The naive estimate is similar to the histogram approach except that rather than dividing the axis up into successive bins, each bin of width *h* is centred on each data point. The density as each $\mathbf{x}_n$ is defined as

$$f(\mathbf{x}_n) = \frac{n_{g_x}}{n} \qquad \text{a2.1}$$

The naive estimate has the advantage of not biasing the estimate with the positions of the bins however it is slightly more computationally expensive. First, it would require all patterns within a distance of *h* to the hypercylinder to be found as well in order to prevent the density along the edges of the hypercylinder from being underestimated. This will also require an additional step to test the distance of patterns from the centroids at the ends of the cylinders. On top of this, the number of bins increases by a factor of *m*.

A2.3 The Kernel Estimator

The kernel estimator can be thought of as a generalisation if the naive estimate. If one were to set *h* such that it included all of the observations in the sample and weight the contribution of each pattern to the density estimate at **x** based on its distance from **x**, one has the kernel estimate of *f(***x***)*. A wide variety of weighting functions can be used. The Gaussian is a common example:

$$\frac{1}{\sqrt{2\Pi}} e^{-\frac{\|\mathbf{x}-\mathbf{x}_n\|^2}{2\sigma}} \qquad \text{a2.2}$$

where σ is a smoothing parameter which operates in much the same way as the number of bins in histograms and the naive estimate. Too large a value of σ will smooth out even local effects while too small a value will enable small and perhaps trivial local conditions to significantly influence the density estimate at that point.

The kernel estimate has the drawback of requiring calculation of the distance between each of the patterns in the data set significantly increasing the computational cost. Although theoretically the entire data set should be used when used the kernel estimator for each hypercylinder, only a subset is needed that are found near the edges. These can be found by only including those observations whose distance to the edge of the hypercylinder is small enough to effect the output of eq. (a2.2). This would help reduce the computational load to some degree. However, once this subset of patterns has been found, the weighted interpattern distance of all patterns in the subset would still have to be calculated.

A2.4 The Nearest Neighbour Estimate

In the nearest neighbour estimate (NNE) that density at **x** is measured as a function of the distance of the *gth*-nearest neighbour to **x**. is defined as:

$$f(\mathbf{x}) = (g - 1)/(2md_g(\mathbf{x})) \qquad \text{a2.3}$$

where $d_k(\mathbf{x})$ is the distance from **x** to its *kth* nearest neighbour.

In highly dense areas $d_g(\mathbf{x})$ will be smaller, while in low density areas $d_g(\mathbf{x})$ will be larger. Therefore the density at **x** is proportional the distance between **x** and its *gth* nearest neighbour. Ultimately, the overall degree of smoothing is dependant on *g*, but the method does allow some adjustment to the smoothing to be determined by the local density. The main computational cost of this approach is that for each hypercylinder, the calculation of each pattern's *g*-nearest neighbours is required which for large data sets may be costly.

A2.5 Summary

The naive, NNE and kernel estimators are significantly more expensive than the histogram estimator. Moreover, the above discussion of the computational costs of both the NN and kernel estimators does not include the determination of the size of smoothing parameter (σ for the kernel estimator and *g* for the NN). The histogram approach is more than adequate for the task, translates easily into the UDD and ADD distance measures, is computationally efficient a relatively simple to implement.

However, note that although the other three estimators have larger computational overheads, they directly calculate the destiny of each observation in the intercluster area, which is more parsimonious with Hartigan's definition of a naturalcluster than histograms. However, because of the higher complexity and CPU costs, the histogram method is the only density estimation technique empirically examined in this thesis.

A3: Sample Code

This appendix contains the C code used to evaluate the hybrid clustering methods. It also includes sample code for the partitioning algorithms used in chapter 6. The following is a summary of key functions included the code listings below:

Data generation:

set_centres( )	Sets cluster centres (Chapter 6).
create_data( )	Data set creation driver.
add_noise( )	Adds noise to data.
gauss()	Gaussian cluster structure.
conic ( )	Conic shaped cluster generator.
third_ring( )	Generates a 120 degree arc shaped cluster.
t_third_ring( )	Generated a 240 degree arc shaped cluster.
ring( )	Generates a ring shaped cluster.
line( )	Generates a linear shaped cluster.

APC:

apc( )	Main APC driver.
get_sd( )	Returns standard deviation of clusters.
check_density( )	Returns intercluster density
near_line_seg( )	Returns distance of observation from hypercylinder axis.
adj( )	Determines neighbouring clusters.

Other clustering algorithms:

k_means( )	K-means.
moving( )	The moving method
kohonen( )	WTANN and KSONN.
slink( )	Single linkage.
alink( )	Average linkage.
clink( )	Centroid linkage.
ward( )	Ward's method.
wong( )	Wong's hybrid method

```
/*****************************************************************************

                    HH_MC.C

        Monte Carlo simulator for hybrid clustering algorithms
        Copyright  Eric W. Tyree^xx
        City University
        Northampton Square
        London, EC1V OHB
        United Kingdom

        xx - gasdev(), ran1() and ran_int() random number routines were adapted from:
        Press, W. H., Teukolsky,  S. A., Vetterling,  W. T., and Flannery B. P. (1995)
        Numerical Recipes in C, Second Ed., Cambridge University Press, Cambridge

*****************************************************************************/


#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <dos.h>

#define RUNS 30                   /* number  of runs */
#define MAXPATS 1500  /* max number of patterns */
#define MAXVARS 2                 /* max number of variables */
#define MAXNOISE 4                /* max noise type */
#define MAXCLUSTERS 20            /* max clusters */
#define EPOCHS 100                /* SONN learning epochs */
#define L_RATE 0.2                /* SONN learing rate */
#define MAXSEGMENTS 100            /* max number of  bins in int.clust. histogram */
#define MIN_SEED 0.0              /*(double)vars/40.0*/
#define MINWINS 38                /* minumum patterns needed in each PP cluster */
#define DEBUG 0                   /* 1 = print debug statements */
#define PIE 3.14159265359

/**** #def's for random number generators  see Press et al (1995) ****/
#define IA 16807
#define IM 2147483647
#define AM (1.0/IM)
#define IQ 127773
#define IR 2836
#define NTAB 32
#define NDIV (1+(IM-1)/NTAB)
#define EPS 1.2e-7
#define RNMX (1.0-EPS)
/******************************/

double SEG_WIDTH =1.0;
int SD_TYPE = 0;
int ADD = 1;

void re_set(int init_parts, int dimension, int num_pats);                    /*data set creation routines */
void shuffle_data(int pats);                                                 /* randomizes order of data*/
int smooth(int num_pats, int num_clusts);              /* returns remainder of num_clusts/num_pats */
double ran1(void);                                     /* returns a uniform random num. from 0.0 to 1.0 */
double noise(double off_set);                          /* same as above but from 0.0 to off_set */
int ran_int(int num);                                  /* returns a random number from 0 to num*/
double gasdev(void);                                   /* returns Gaussian deviate of m = 0, sd = 1 */

        /* data creation driver*/
void create_data(int num_pats,int dimension,int data_type,int num_clusts, double disp, double **mean);

        /* sets cluster centres */
void set_centres(int num_clusts, int vars, double **mean);

        /* adds noise to data */
void add_noise(int num_pats,int dimension,int noise_type, double disp);

        /* replaces vars variables with random uniform noise in the range of min to max */
void uniform(int vars, int samples, int start, double max, double min);

        /* simulated data cluster shape creation routines */
void guass(long clust_num,double off_set,int vars, samples, int start, double *mean, double *variance);

void conic(int samples, int clust_num,double off_set,int start, vars,double *centre, double disp);
void third_ring(int samples, int clust_num,double off_set,int start, vars, double r,double width,double
        *centre);
void t_third_ring(int samples, int clust_num,double off_set,int start, vars, double r,double width,double
        *centre, double start_sec);
void ring(int samples, int clust_num,double off_set,int start,int vars, double r,double width,double
        *centre);
void line(int samples, int clust_num,double off_set,int start,int vars,double *centre, double disp, double
        yint);

/* partitioning clustering algorithms */
int k_means(int pats, int vars, int clusters, int converge);
int moving(int pats, int vars, int clusters, int converge);
void kohonen(int num_clusts,int vars, int pats, int epochs, double l_rate, int nbs,int top);
int get_partition(int pats,int vars,int num_clusts);


        /* APC */
double get_sd(double wc,double wg);
double check_density(int clust2,int clust1,int vars, int pats, double width);
double near_line_seg(double B, int pat,int clust1, int clust2,int vars,double width);
int adj(int pats, int vars, int stop, int k)

/* hierarchical clustering algorithms */
void slink(int pats, int vars, int stop, int k); /* single linkage */
void alink(int pats, int vars, int stop, int k); /* average linkage */
void clink(int pats, int vars, int stop, int k); /* centroid linkage */
void ward(int pats, int vars, int stop, int k); /* Ward's technique */
int wong(int pats, int vars, int stop, int k);     /* Wong's hybrid method
```

```
                                              returns 0 if "stop" clusters
                                              cannot be found */

double get_adj_rand(int num_clusts, int pats /* get rand statistic */
double get_mem(int num_clusts, int vars, int pats);  /* get cluster memberships
                                                       of each pattern */
/* Global variables */

double centroids[MAXCLUSTERS][MAXVARS];   /* cluster centroids */
double d_matrix[MAXCLUSTERS][MAXCLUSTERS];       /* distance matrix */

double **data;    /* data vectors */
int *cat;         /* actual cluster membership */
int *mem;         /* clstr. memersips of observations found via clustering algorithms */

double **mean;  /* holds cluster centres */
int orient[MAXCLUSTERS],seedx[MAXCLUSTERS],seedy[MAXCLUSTERS];
double y_int[MAXCLUSTERS];
int c_mem[MAXCLUSTERS];       /* holds cluster membership of each pattern */
long *idum;
double temp_cents[MAXCLUSTERS][MAXVARS];

 /* these hold various stats
  0 = k-means, 1 = moving, 2 = kohonen, 3 = unsupervised */

double r[5][RUNS];                /* holds adjusted rand results */
double r_mean[4],r_var[4];
double con[4][RUNS];              /* holds convergence error results */
double con_mean[4],con_var[4];
int iter[2][RUNS];
double iter_m[2],iter_v[2];       /* holds number of iterations */
FILE *fp,*sas;


void main(void)
{
        char file[50];
        int run;
        int dimension                 /* number of variables */
         num_pats = MAXPATS,          /* data set size */
        noise_type,                   /* type of noise condition */
        data_set,                     /* data set index */
        num_clusts;                   /* number of clusters */
        double disp;                  /* disperison */
        int c,m,p,v,w;
        int init_parts,parts;
        int d,set,noise,clust,var;    /* counters for output to sas */

        data =  (double **)malloc(MAXPATS*sizeof(double *));
         for(c = 0; c < MAXPATS;c++)
                data[c] = (double *)malloc(MAXVARS*sizeof(double));
        mem = (int *)malloc(MAXPATS*sizeof(int));
        cat = (int *)malloc(MAXPATS*sizeof(int));
        mean = (double **)malloc(MAXCLUSTERS*sizeof(double*));
```

```
for(m = 0; m < MAXCLUSTERS; m++)
        mean[m] = (double *)malloc(MAXVARS*sizeof(double));

printf("ENTER SAS OUTPUT FILE\n");
gets(file);

if((sas = fopen(file,"w")) == NULL)
{
        printf("COULD NOT OPEN %s\n",file);
        exit(0);
}

for(data_set = 4; data_set <= 4; data_set++)                    /* data set */
for(dimension = 4;dimension <= 10; dimension += 3)              /* variables */
for(num_clusts = 2; num_clusts <= 2; num_clusts += 2)           /* number of clusters */
{
        set_centres(num_clusts,dimension,mean);                 /* hold cluster centres constant */
        for(noise_type = 0; noise_type <= 4; noise_type++)      /* noise */
        for(disp = 0.025; disp <= 0.0755; disp += 0.025)        /* set dispersion */
        for(parts = 15; parts <= 15; parts += 5)                /* initial number of clusters */
        {
                for(run = 0; run < RUNS; run++) /* repeat */
                {
                        /* re-run same dataset RUNS times */

                        create_data(num_pats,dimension,data_set,num_clusts,disp,mean);
                        shuffle_data(num_pats);
                        add_noise(num_pats,dimension,noise_type,disp);
                        shuffle_data(num_pats);

                        init_parts =  get_partition(num_pats,dimension,init_parts);
                        for(c = 0; c < init_parts; c++)
                                for(v = 0; v < dimension; v++)
                                        temp_cents[c][v] = centroids[c][v];

                        ADD = 0;  /* RUN APC WITH UDD */

                        re_set(init_parts,dimension,num_pats);
                        apc(init_parts, dimension, num_pats, num_clusts);
                        r[0][run] = get_adj_rand(init_parts,num_pats);
                        fprintf(sas,"apc_u %.4lf %d %d %d %d %.3lf

%d\n",r[0][run],data_set,num_clusts,dimension,noise_type,disp,parts);

                        ADD = 1; /* RUN APC WITH ADD */

                        re_set(init_parts,dimension,num_pats);
                        apc(init_parts, dimension, num_pats, num_clusts);
                        r[0][run] = get_adj_rand(init_parts,num_pats);
                        fprintf(sas,"apc_a %.4lf %d %d %d %d %.3lf

%d\n",r[0][run],data_set,num_clusts,dimension,noise_type,disp,parts);

                        /* RUN WONG'S HYBRID METHOD */
```

```
                                        re_set(init_parts,dimension,num_pats);
                                        wong(num_pats,dimension,num_clusts,init_parts);
                                        r[0][run] =  get_adj_rand(init_parts,num_pats);
                                        fprintf(sas,"wong %.4lf %d %d %d %d %.3lf

%d\n",r[0][run],data_set,num_clusts,dimension,noise_type,disp,parts);

                                        /* RUN SINGLE LINKAGE */

                                        re_set(init_parts,dimension,num_pats);
                                        slink(num_pats,dimension,num_clusts,init_parts);
                                        r[0][run] =  get_adj_rand(init_parts,num_pats);
                                        fprintf(sas,"slink %.4lf %d %d %d %d %.3lf

%d\n",r[0][run],data_set,num_clusts,dimension,noise_type,disp,parts);

                                        /* RUN CENTROID LINKAGE */

                                        re_set(init_parts,dimension,num_pats);
                                        clink(num_pats,dimension,num_clusts,init_parts);
                                        r[0][run] = get_adj_rand(init_parts,num_pats);
                                        fprintf(sas,"clink %.4lf %d %d %d %d %.3lf

%d\n",r[0][run],data_set,num_clusts,dimension,noise_type,disp,parts);

                                        /* RUN AVERAGE LINKAGE */

                                        re_set(init_parts,dimension,num_pats);
                                        alink(num_pats,dimension,num_clusts,init_parts);
                                        r[0][run] = get_adj_rand(init_parts,num_pats);
                                        fprintf(sas,"alink %.4lf %d %d %d %d %.3lf

%d\n",r[0][run],data_set,num_clusts,dimension,noise_type,disp,parts);

                                        /* RUN WARD'S METHOD */

                                        re_set(init_parts,dimension,num_pats);
                                        ward(num_pats,dimension,num_clusts,init_parts);
                                        r[0][run] = get_adj_rand(init_parts,num_pats);
                                        fprintf(sas,"ward %.4lf %d %d %d %d %.3lf

%d\n",r[0][run],data_set,num_clusts,dimension,noise_type,disp,parts);

                        } /* end RUNS loop */
            } /* end dispersion loop */
} /* end num_clusts loop */
free(mem);
free(cat);

for(c = 0; c < MAXPATS;c++)
            free(data[c]);
/* fclose(fp);  */
for(m = 0; m < MAXCLUSTERS; m++)
```

```
                free(mean[m]);
        free(mean);

}/* END MAIN ROUTINE */

void re_set(int init_parts, int dimension, int num_pats)
{
        int c,v;
        for(c = 0; c < init_parts; c++)
                for(v = 0; v < dimension; v++)
                        centroids[c][v] = temp_cents[c][v];
        get_mem(init_parts,dimension,num_pats);
}

void set_centres(int num_clusts, int vars, double **mean)
{
        int c,m,y;
        int used[MAXCLUSTERS];
        long temp;
        double delta;

        temp = 0 - (long)time(NULL);    /* seed random number generator */
        idum = &temp;

        /* randomly assign means on each dimension for each cluster to
                ensure maximum separation of means on each dimension*/

        delta = 1.0/(num_clusts + 1);
        for(m = 0; m < vars; m++)
        {
                for(c = 0; c < num_clusts; c++)
                        used[c] = c+1;
                for(c = 0; c < num_clusts; c++)
                {
                        do{
                                y = ran_int(num_clusts);
                        }while(!used[y]);
                        mean[c][m] = used[y]*delta;
                        used[y] = 0;
                }
        }
        for(c = 0; c < num_clusts; c++)
        {
                seedx[c] = ran_int(vars);  /* randomly set orientation */
                do{
                        seedy[c] = ran_int(vars);
                }while(seedy[c] == seedx[c]);

                /* randomly choose orientation of density gradient -
                        this is used in the conic clusters */
                if(ran1() >= 0.5)
                        orient[c] = 1;
                else
                        orient[c] = 0;
                y_int[c] = ran1();
        }
}

double get_mem(int num_clusts, int vars, int pats)
{
        int c,v,p;
        double dist,min_dist,error;

        /* get the cluster memberships of individual patterns as determined by the
                partitioning algorithm. A pattern is a member of the cluster who's centroid
                it is closest to */

        error = 0.0;
        for(p = 0; p < pats; p++)
        {
                min_dist = 99999999.0;
                for(c = 0; c < num_clusts; c++)
                {
                        dist = 0.0;
                        for(v = 0; v < vars; v++)
                                dist += (data[p][v] - centroids[c][v])*(data[p][v] - centroids[c][v]);
                        if(dist < min_dist)
                        {
                                mem[p] = c;
                                min_dist = dist;
                        }
                        error += min_dist;
                } /* end cluster loop */
        } /* end pattern loop */
        return(error/(double)pats);
}

double get_adj_rand(int num_clusts, int pats)
{
        /* calculates adusted RAND coefficient */

        int i,j,p;
        double crand,ni2,nj2,nij2,n2;

        /* stats for adjusted RAND */
        double nij[MAXCLUSTERS][MAXCLUSTERS];
        double ni[MAXCLUSTERS];
        double nj[MAXCLUSTERS];

        n2 = (double)pats*((double)pats - 1.0)/2.0;

        for(i = 0; i < num_clusts; i++)
        {
                nj[i] = 0.0;
                ni[i] = 0.0;
                for(j = 0; j < num_clusts; j++)
                        nij[i][j] = 0.0;
        }
        for(p = 0; p < pats; p++)
                nij[ mem[p] ][ cat[p] ] += 1.0;
                for(i = 0; i < num_clusts; i++)
                        for(j = 0; j < num_clusts; j++)
                        {
                                ni[i] += nij[i][j];
                                nj[j] += nij[i][j];
                        }

                ni2 = 0.0;
                nj2 = 0.0;
                nij2 = 0.0;
                for(i = 0; i < num_clusts; i++)
                {
                        if(ni[i] > 1.0)
                        ni2 += ni[i]*(ni[i] - 1.0)/2.0;
                        if(nj[i] > 1.0)
                                nj2 += nj[i]*(nj[i] - 1.0)/2.0;
                        for(j = 0; j < num_clusts; j++)
                        {
                                if(nij[i][j] > 1.0)
                                nij2 += nij[i][j]*(nij[i][j] - 1.0)/2.0;
                        }
        }
        crand = (nij2 - (ni2*nj2/n2))/((ni2/2.0) + (nj2/2.0) - ((ni2*nj2)/n2));
        return(crand);
}

int k_means(int pats, int vars, int clusters, int converge)
{
        /* This procedure implements Forgy's k-means algorithm */

        int c,v,closest,iteration,pt;
        int p,num_change;
        int membership[MAXPATS];
        double new_centroids[MAXCLUSTERS][MAXVARS];
        int num_members[MAXCLUSTERS];
        long temp;
        double dist, min_dist,*dtemp;
        int p1,p2,tempp,g,ctemp,mtemp;
        long temp;

        for(p = 0; p < pats;p++)
                membership[p] = -1;
        /* seed centroids and initialize */
        temp = 0 - (long)time(NULL);    /* seed random number generator */
        idum = &temp;

        /*  initialize*/
        for(c = 0; c < clusters; c++)
                {
                        for(v = 0;v < vars; v++)
                                new_centroids[c][v] = 0.0;
                num_members[c] = 0;
        }
num_change = converge + 1;
iteration = 0;

while(num_change > converge)
{
        num_change = 0;
        /****** data shuffle ********/
        temp = 0 - (long)time(NULL);    /* seed random number generator */
        idum = &temp;

        for(p = 0; p < pats;p++)
        {
                g = ran_int(pats);
                dtemp = data[p];
                ctemp = cat[p];
                mtemp = membership[p];
                data[p] = data[g];
                cat[p] = cat[g];
                membership[p] = membership[g];
                data[g] = dtemp;
                cat[g] = ctemp;
                membership[g] = mtemp;
        }

         /*************************/
        for(p = 0; p < pats; p++)
        {
                min_dist = 1000000.0;
                closest = 0;
                for(c = 0; c < clusters;c++)  /* find cluster centroid closest to pattern */
                {
                        dist = 0.0;
                        for(v = 0; v < vars; v++)
                                dist += (centroids[c][v] - data[p][v])*( centroids[c][v] -
                                                        data[p][v]);
                        if(dist < min_dist)
                        {
                                min_dist = dist;
                                closest = c;
                        }
                }
                num_members[closest]++;
                for(v = 0; v < vars; v++)
                        new_centroids[closest][v] += data[p][v];
                if(closest != membership[p])
                        num_change++;
                membership[p] = closest;
        } /* end pattern loop*/
        for(c = 0; c < clusters; c++)  /* reclaculate cluster centroids */
        {
                for(v = 0;v < vars; v++)
                {
```

```
                                        if(num_members[c] == 0);
                                        else
                                                centroids[c][v] =

                new_centroids[c][v]/(double)num_members[c];
                                        new_centroids[c][v] = 0.0;
                                }
                                num_members[c] = 0;
                        }
                        iteration++;
                } /* end iteration loop */
                return(iteration);
}

int seed(int pats, int clusters, int vars)
{
                int c,cs,v,reject,g;
                double dist;

                /* this procedure randomly select patterns to serve as seeds. The only constraint is
                        that they be MIN_DIST distant from each other. */
                g = ran_int(pats);  /* randomly allocate a pattern as the first seed */
                for(v = 0; v < vars; v++)
                        centroids[0][v] = data[g][v];

                for(cs = 1; cs < clusters;cs++) /* do the same for rest of seeds */
                {
                        reject = 0;
                        do{
                                g = ran_int(pats);
                                for(c = 0; c < cs; c++)  /* make sure seed is MIN_SEED dist from all others */
                                        {
                                                dist = 0.0;
                                                for(v = 0; v < vars; v++)
                                                        dist += (data[g][v] - centroids[c][v])*(data[g][v] - c
                                                                        centroids[c][v]);
                                                if(sqrt(dist) < MIN_SEED)
                                                {
                                                        printf("REJECT\n");
                                                        reject = 1;
                                                }
                                }
                        }while(reject);
                        for(v = 0; v < vars; v++)
                                centroids[cs][v] = data[g][v];
                }
                return(1);
}

int moving(int pats, int vars, int clusters, int converge)
{

                /* This procedure implements the moving method. Only the best possible
                        move is kept */
```

```
int c,v,closest,iteration,pt;
int p,num_change,min_index;
int membership[MAXPATS];
double new_centroids[MAXCLUSTERS][MAXVARS];
int num_members[MAXCLUSTERS];
double dist, min_dist, error, test_error,*dtemp;
int order[MAXPATS];
int p1,p2,tempp,g,ctemp,mtemp;
long temp;

temp = 0 - (long)time(NULL);    /* seed random number generator */
idum = &temp;

for(c = 0; c < clusters; c++)  /* initialize*/
{
        num_members[c] = 0;
        for(v = 0; v < vars; v++)
                new_centroids[c][v] = 0.0;
}

/* initialize initial clusters */
for(p = 0; p < pats; p++)
{
        min_dist = 10000000.0;
        min_index = 0;
        for(c = 0; c < clusters; c++)
        {
                dist = 0.0;
                for(v = 0; v < vars; v++)
                        dist += (data[p][v] - centroids[c][v])*(data[p][v] - centroids[c][v]);
                if(dist < min_dist)
                {
                        min_dist = dist;
                        min_index = c;
                }
        }
        membership[p] = min_index;
        num_members[min_index]++;
        for(v = 0; v < vars; v++)
                new_centroids[min_index][v] += data[p][v];
}
for(c = 0; c < clusters; c++)
{
        for(v = 0; v <vars; v++)
        {
                if(num_members[c] == 0);
                else
                        centroids[c][v] = new_centroids[c][v]/num_members[c];
        }
}
num_change = converge + 1;
iteration = 0;
while(num_change > converge)
```

```
{
        num_change = 0;
        /****** data shuffle ********/
        temp = 0 - (long)time(NULL);    /* seed random number generator */
        idum = &temp;

        for(p = 0; p < pats;p++)
        {
                g = ran_int(pats);
                dtemp = data[p];
                ctemp = cat[p];
                mtemp = membership[p];
                data[p] = data[g];
                cat[p] = cat[g];
                membership[p] = membership[g];
                data[g] = dtemp;
                cat[g] = ctemp;
                membership[g] = mtemp;
        }
        for(p = 0; p < pats; p++)
        {
                test_error = 0.0;
                        /* calculate the cost of removing pat from its current cluster */
                if(num_members[membership[p]] == 1);
                else
                {
                        for(v = 0; v < vars; v++)
                                test_error += (centroids[membership[p]][v] - data[p][v])*
                                                (centroids[membership[p]][v] - data[p][v]);
                        test_error = (num_members[membership[p]]*test_error)/
                                        (num_members[membership[p]] - 1);
        }
        min_index = membership[p];

        /* calculate the cost of adding pat to each of the remaining clusters - find best */
        for(c = 0; c < clusters; c++)
                if(c != membership[p])
                {
                        error = 0.0;
                        for(v = 0; v < vars; v++)
                                error += (centroids[c][v] - data[p][v])*
                                        (centroids[c][v] - data[p][v]);
                        error = (num_members[c]*error)/(num_members[c] + 1);

                        if(error < test_error)  /**** find best move ****/
                        {
                                test_error = error;
                                min_index = c;


                        }
                }

                if(min_index != membership[p])
                {/* move pattern and adjust centoids, membership array etc. */
```

```
                                        for(v = 0; v < vars; v++)
                                        {
                                                centroids[membership[p]][v] =
                                                        ((num_members[membership[p]]*
                                                                centroids[membership[p]][v]) -

        data[p][v])/(num_members[membership[p]] - 1);
                                                centroids[min_index][v] =
                                                        ((num_members[min_index]*
                                                                centroids[min_index][v]) +

        data[p][v])/(num_members[min_index] + 1);
                                                }
                                                num_members[membership[p]]--;
                                                num_members[min_index]++;
                                                membership[p] = min_index;
                                                num_change++;
                                        }
                                }/*** end pattern loop ***/
                                iteration++;
                        } /** end iteration ***/
        return(iteration);
        } /*** END MOVING METHOD ***/

void kohonen(int num_clusts,int vars, int pats, int epochs, double l_rate, int nbs,int top)
{

        /* This procedure implements both a Kohonen and standard unsupervised networks
           top = 1 (Kohonen), top = 0 (standard unsupervised) */

        int p,e,pattern,pt;
        long temp;
        int v,c,closest,start, finish;
        double *means,init_learn,dist, c_dist,init_nbs;

        init_nbs = floor(0.66*(double)num_clusts);
        nbs = (int)floor(0.66*(double)num_clusts);
        init_learn = l_rate;
        means = (double *)malloc(vars*sizeof(double));

        /**** randomize initial weights ****/

        for(v = 0; v < vars; v++)
                means[v] = 0.0;         /*** first calc. means ***/

        for(p = 0; p < pats; p++)
                for(v = 0; v < vars;v++)
                        means[v] += data[p][v];
        for(v = 0; v < vars; v++)
                means[v] /= (double)pats;

        temp = 0 - (long)time(NULL);    /* seed random number generator */
        idum = &temp;
        for(c = 0; c < num_clusts; c++)
```

```
                for(v = 0; v < vars; v++)
                        centroids[c][v] = means[v] + ((0.1*(double)ran_int(32000)/
                                                                (double)32000) - 0.05);
        free(means);

        /*printf("epochs %ld\n",epochs);*/
        /********** train network ***********/

        for(e = epochs; e > 0; e--)
        {
                shuffle_data(pats);
                for(p = 0; p < pats; p++)
                {
                        pattern = ran_int((int)pats); /* randomly choose a pattern */
                        c_dist = 999999999999.0;
                        closest = 0;
                        for(c = 0; c < num_clusts;c++) /* find closest cluster */
                        {
                                dist = 0.0;
                                for(v = 0; v < vars; v++)
                                        dist += fabs(data[pattern][v] - centroids[c][v]);
                                if(dist <  c_dist)
                                {
                                        c_dist = dist;
                                        closest = c;
                                }
                        }
                        /**** update weights of closest cluster (unit) ****/
                        if(top) /* use Kohonen's topological update rule */
                        {
                                if(closest - nbs < 0)
                                        start = 0;
                                else
                                        start = closest - nbs;
                                if(closest + nbs > num_clusts - 1)
                                        finish = num_clusts - 1;
                                else
                                        finish = closest - nbs;
                                for(c = start; c <= finish; c++)
                                        for(v = 0; v < vars; v++)
                                                centroids[c][v] += l_rate*(data[pattern][v] -
                                                                        centroids[c][v]);
                        }
                        else /* update only the winning cluster (unit) */
                                for(v = 0; v < vars; v++)
                        centroids[closest][v] +=l_rate*(data[pattern][v] - centroids[closest][v]);
                } /*** end pattern loop ***/
                l_rate = l_rate - init_learn/(double)epochs;
                 nbs = (int)floor(init_nbs*(double)e/(double)epochs);
        }
}

void create_data(int num_pats,int dimension,int data_type, int num_clusts,
        double disp, double **mean)
```

```
{
          double off_set = 0.001;
          double *variance;
          int m,pats,extra,c,shape,p;
          long temp;

          /* This procedure drives the procedures that generate the simulated
                    data. */

          variance = (double *)malloc(dimension*sizeof(double));
          temp = 0 - (long)time(NULL);    /* seed random number generator */
          idum = &temp;

          /* calculate the number of patterns in each cluster,
          save any remainder in extra variable. Last cluster to be produced
          will have pats/num_clusts + extra patterns in it */
          pats = smooth(num_pats,num_clusts);
          xtra = num_pats%num_clusts;

          for(m = 0; m < dimension; m++)  /* set dispersion */
                    variance[m] = disp;

          if(data_type == 0) /* Gaussian clusters */
          {

                    /* generate all but last cluster */
                    for(c = 0; c < num_clusts - 1; c++)
                    guass(/*clust_num,off_set,vars,samples,start,*mean,*variance*/
                              c,off_set,dimension,pats,c*pats,mean[c],variance);

                    /* generate last cluster to include patterns that are left over
                              from the the division of num_pats/num_clusts */
                    if(DEBUG) printf("cnum %d, dim %d samps %d start %d \n",num_clusts - 1,
                              dimension,pats + extra,pats*(num_clusts - 1));
                    guass(/*clust_num,off_set,vars,samples,start,*mean,*variance*/
                              num_clusts - 1,off_set,dimension,pats + extra,
                              pats*(num_clusts - 1),mean[num_clusts-1],variance);
          }

          else if(data_type == 1)  /* third of ring shaped clusters*/
          {
                    /* set centres as with Gaussian using mean to hold centre coordinates */
                    /* generate all but last cluster */

                    for(c = 0; c < num_clusts - 1; c++)
                              third_ring(/*samples,clust_num,off_set,start,vars,r,width*/
                                        pats,c,off_set,pats*c,dimension,0.2,disp,mean[c]);

                    /* create remaining cluster */

                    third_ring(/*samples,clust_num,off_set,start,vars,r,disp, centre*/
                                        pats+extra,num_clusts - 1,off_set,pats*(num_clusts - 1),dimension,
                                        0.1,disp,mean[num_clusts - 1]);
```

```
}
else if(data_type == 2) /* conic shaped clusters*/
{

        for(c = 0; c < num_clusts - 1;c++)
        conic(/*samples, clust_num, off_set,start, vars,mean,disp*/
                pats,c,off_set,pats*c,dimension,mean[c],disp);

        conic(/*samples, clust_num, off_set,start, vars,mean,disp*/
                pats+extra,num_clusts - 1,off_set,pats*(num_clusts - 1),
                dimension,mean[num_clusts - 1],disp);
}
else if(data_type == 3) /* random mixture of shapes */
{
        for(c = 0; c < num_clusts - 1;c++)
        {
                shape = ran_int(3);
                if(shape == 0)
                        guass(/*clust_num,off_set,vars,samples,start,*mean,*variance*/
                                c,off_set,dimension,pats,c*pats,mean[c],variance);
                else if(shape == 1)
                        third_ring(/*samples,clust_num,off_set,start,vars,r,width*/
                                pats,c,off_set,pats*c,dimension,0.15,disp,mean[c]);
                else if(shape == 2)
                        conic(/*samples, clust_num, off_set,start, vars,mean,disp*/
                        pats,c,off_set,pats*c,dimension,mean[c],disp);
                else
                {
                        printf("SHAPE INDEX ERROR\n");
                        exit(0);
                }
        }
                shape = ran_int(2);
        if(shape == 0)
                guass(/*clust_num,off_set,vars,samples,start,*mean,*variance*/
                                num_clusts - 1,off_set,dimension,pats + extra,
                                pats*(num_clusts - 1),mean[num_clusts-1],variance);
        else if(shape == 1)
        third_ring(/*samples,clust_num,off_set,start,vars,r,disp, centre*/
                pats+extra,num_clusts - 1,off_set,pats*(num_clusts -
                1),dimension,0.1,disp,mean[num_clusts - 1]);
        else if(shape == 2)
                conic(/*samples, clust_num, off_set,start, vars,mean,disp*/
                        pats+extra,num_clusts - 1,off_set,pats*(num_clusts - 1),
                        dimension,mean[num_clusts - 1],disp);
        else
        {
                printf("SHAPE INDEX ERROR\n");
                exit(0);
        }
}
else if(data_type == 4)  /* Concentric clusters (ring around a Gaussian blob) */
{
        num_clusts = 4;
        pats = smooth(num_pats,num_clusts);
        extra = num_pats%num_clusts;
        for(c = 0; c < dimension;c++)
                        mean[0][c] = 0.5;
        ring(/*samples,clust_num,off_set,start,vars,r,width*/
                        3*pats,0,off_set,0,dimension,0.35,disp,mean[0]);
        printf("GOT RING\n");
        guass(/*clust_num,off_set,vars,samples,start,*mean,*variance*/
                1,off_set,dimension,pats + extra,pats*3,mean[0],variance);
        printf("GOT GAUSS\n");
}
else if(data_type == 5)  /* three parallel lines */
{

        num_clusts = 3;
        pats = smooth(num_pats,num_clusts);
        extra = num_pats%num_clusts;
        for(c = 0; c < 2;c++)
        {
                mean[0][c] = 0.3;
                mean[1][c] = 0.5;
                mean[2][c] = 0.7;
        }
        for(c = 2; c < dimension;c++)
        {
                mean[0][c] = 0.0;
                mean[1][c] = 0.0;
                mean[2][c] = 0.0;
        }
        line(/*samples, clust_num, off_set,start, vars,mean,disp,yint*/
                pats,0,off_set,0,dimension,mean[0],disp,0.3);

        line(/*samples, clust_num, off_set,start, vars,mean,disp,yint*/
                pats,1,off_set,pats,dimension,mean[1],disp,0.5);

        ine(/*samples, clust_num, off_set,start, vars,mean,disp,yint*/
                pats+extra,2,off_set,pats*2,dimension,mean[2],disp,0.7);
}
else if(data_type == 6)  /* EXIT - THIS ONE IS NO LONGER USED */
{
        printf("No cluster type 6 - exiting ... ");
        exit(2);
}
else if(data_type == 7) /* Interlocking clusters */
{

        num_clusts = 2;
        pats = smooth(num_pats,num_clusts);
        extra = num_pats%num_clusts;

        mean[0][0] = 0.4;
        mean[0][1] = 0.5;
        mean[1][0] = 0.8;
        mean[1][1] = 0.5;
```

```
			for(c = 2; c < dimension;c++)
			{
				mean[0][c] = 0.0;
				mean[1][c] = 0.0;
			}
			t_third_ring(/*samples,clust_num,off_set,start,vars,r,width,centre,start_sec*/
				pats,0,off_set,0,dimension,0.4,disp,mean[0],0.0);

			printf("Got SECNd RING\n");*/
			t_third_ring(/*samples,clust_num,off_set,start,vars,r,width,centre,start_sec*/
				pats+extra,1,off_set,pats,dimension,0.4,disp,mean[1],1.0);
		}
		else if(data_type == 8) /* Winding cluster with two Gaussians */
		{
			num_clusts = 6;
			pats = smooth(num_pats,num_clusts);
			extra = num_pats%num_clusts;
			mean[0][0] = 0.5;
			mean[0][1] = 0.5;
			mean[1][0] = 1.2;
			mean[1][1] = 0.5;

			for(c = 2; c < dimension;c++)
			{
				mean[0][c] = 0.0;
				mean[1][c] = 0.0;
			}
			/*clust_num vars pats start disp*/
			t_third_ring(/*samples,clust_num,off_set,start,vars,r,width,centre,start_sec*/
				2*pats,0,off_set,0,dimension,0.35,disp,mean[0],0.0);

			t_third_ring(/*samples,clust_num,off_set,start,vars,r,width,centre,start_sec*/
				2*pats,0,off_set,2*pats,dimension,0.35,disp,mean[1],1.0);

			guass(/*clust_num,off_set,vars,samples,start,*mean,*variance*/
				1,off_set,dimension,pats,4*pats,mean[0],variance);

			guass(/*clust_num,off_set,vars,samples,start,*mean,*variance*/
				2,off_set,dimension,pats+extra,5*pats,mean[1],variance);
		}

		else
		{
			printf("data_type: %d unrecognized\n",data_type);
			exit(1);
		}

	free(variance);
	} /*END DATA GENERATION */

	void add_noise(int num_pats,int dimension,int noise_type, double disp)
	{
		int p,g,m;
		long temp;
```

```
double gd;
double min[MAXVARS];
double max[MAXVARS];

/* This procedure adds noise to the generated data */

temp = 0 - (long)time(NULL);        /* seed random number generator */
idum = &temp;
if(noise_type > 2)                  /* find range on each dimension for uniform noise */
{
        for(m = 0; m < dimension; m++)
        {
                min[m] = 0.0;  /* produce noise in range of 0 - 1 unless a dimesnion */
                max[m] = 1.0;  /* has a greater range. In this case set range to range of data */
        }

        /* find max and mins on each variable */

        for(p = 0; p < num_pats; p++)
                for(m = 0; m < dimension; m++)
                {
                        if(data[p][m] > max[m])
                                max[m] = data[p][m];
                        if(data[p][m] < min[m])
                                min[m] = data[p][m];
                }
}
/* noise_type  = 0: no noise */

if(noise_type == 0)
        return;

/* noise_type = 1 - 2: randomly perturb 50% of interpoint distances */
else if(noise_type == 1)
        for(p = 0; p <= (int)((double)num_pats*0.5); p++)
                for(m = 0; m < dimension; m++)
                        data[p][m] += NLOW*disp*gasdev();

else if(noise_type == 2)
        for(p = 0; p <= (int)((double)num_pats*0.5); p++)
                for(m = 0; m < dimension; m++)
                        data[p][m] += NHIGH*disp*gasdev();

else if(noise_type == 3) /* replace 25% of data with uniform noise */
        for(p = 0; p <= (int)((double)num_pats*ULOW); p++)
                for(m = 0; m < dimension; m++)
                        data[p][m] = (ran1()*(max[m] - min[m])) + min[m];

else if(noise_type == 4) /* replace 50% of data with uniform noise*/
        for(p = 0; p <= (int)((double)num_pats*UHIGH); p++)
                for(m = 0; m < dimension; m++)
                        data[p][m] = (ran1()*(max[m] - min[m])) + min[m];

else
```

```
        {
                printf("NOISE TYPE %d NOT RECOGNIZED\n",noise_type);
                exit(1);
        }
        return;
}

void shuffle_data(int pats)  /* randomly shuffle the order of the data patterns */
{
        int p,ctemp,g;
        double *dtemp;
        long temp;

        temp = 0 - (long)time(NULL);   /* seed random number generator */
        idum = &temp;
        for(p = 0; p < pats;p++)
        {
                g = ran_int(pats);
                dtemp = data[p];
                ctemp = cat[p];
                data[p] = data[g];
                cat[p] = cat[g];
                data[g] = dtemp;
                cat[g] = ctemp;

        }
}

double ran1(void)
{ /* return uniform random deviate in range of 0.0 and 1.0 */

        int j;
        long k;
        static long iy = 0;
        static long iv[NTAB];
        double temp;

        if(*idum <= 0 || !iy)  /* intialize */
        {
                if(-(*idum) < 1)  /* prevent idum = 0; */
                        *idum = 1;
                else
                        *idum = -(*idum);
                for(j = NTAB+7;j >= 0 ; j--)   /* load shuffle table after 8 warm ups */
                {
                        k = (*idum)/IQ;
                        *idum = IA*(*idum - k*IQ) - IR*k;
                        if(*idum < 0)
                                *idum += IM;
                        if(j < NTAB)
                                iv[j] = *idum;
                }
        }
        k = (*idum)/IQ;
        *idum = IA*(*idum - k*IQ) - IR*k;
        if(*idum < 0)
                *idum += IM;
        j = iy/NDIV;
        iy = iv[j];
        iv[j] = *idum;
        if((temp = AM*iy) > RNMX)
                return(RNMX);
        else
                return(temp);
} /*** END RAN1 ***/

double noise(double off_set)
{
        return((0.0 - off_set/2.0) + off_set*ran1());
}

void guass(long clust_num,double off_set,int vars,
        int samples, int start, double *mean, double *variance)
{
        int m,s;   /* returns a normally distibuted number */

        for(s = start; s < start+samples;s++)
        {
                for(m = 0; m < vars; m++)
                data[s][m] = ((gasdev()*variance[m]) + mean[m]) + noise(off_set);
                        cat[s] = clust_num;
        }
}

double gasdev(void)
{
        static int iset = 0;
        static double gset;
        double fac,rsq,v1,v2;

        if(iset == 0)
        {
                do{
                        v1 = 2.0*ran1() - 1.0;
                        v2 = 2.0*ran1() - 1.0;
                        rsq = v1*v1+v2*v2;
                }while(rsq >= 1.0 || rsq == 0.0);
                fac = sqrt(-2.0*log(rsq)/rsq);
                gset = v1*fac;
                iset = 1;
                return(v2*fac);
        }
        else
        {
                iset = 0;
                return(gset);
        }
} /**** END GSDEV ****/
```

```
void uniform(int vars, int samples, int start, double max, double min)
{ /* replaces vars variables with random uniform noise in the range of min to max */

        int m,s;

        for(s = start; s < samples+start;s++)
                for(m = 0; m < vars;m++)
                        data[s][m] = (max - min)*ran1() + min;

} /*** END UNIFORM ***/

void conic(int samples, int clust_num,double off_set,int start,
          int vars,double *centre, double disp)
{
        /* genrates a conic section by taking a line whose midpoint
                 is defined by the centre array of lenth 0.3
                and increasing the dispersion from one end to the other */

        int tot,v,y,plane;
        double x,output,slope,end1,end2,width;

        /* randomly choose a plane to put line on */
        /* plane = ran_int(vars);
        do{
                y = ran_int(vars);
        }while(y == plane); */

        plane = seedx[clust_num];
        y = seedy[clust_num];

        end1 = centre[plane] - 0.14;
        end2 = centre[plane] + 0.14;
        /* randomly choose a slope in range of 0.0 - 1.0 */

        slope = 0.0;
        tot = start;
        while(tot < start+samples)
        {
                /* choose a point on the line */
                x = (ran1()*(end2 - end1)) + end1;
                output = slope*x + y_int[clust_num];
                /*calculate "width" of line at the point*/
                if(orient)
                        width = (disp/4.0)+(((x - end1)/0.28)*disp);
                else
                        width = (disp/4.0)+(((end2 - x)/0.28)*disp);
                data[tot][plane] = x;
                data[tot][y] = output + (ran1()*width);

                /* create tubular region around the line with radius = width*/
                for(v = 0; v < vars; v++)
                        if((v != plane) && (v != y))
                                data[tot][v] = output + (ran1()*width);
                cat[tot] = clust_num;
                tot++;

        } /* end while samples */
}

void line(int samples, int clust_num,double off_set,int start,
          int vars,double *centre, double disp, double yint)
{
        /* genrates  a line whose midpoint
                is defined by the centre */

        int tot,v,y,plane;
        double x,output,slope,end1,end2,width;

        /* randomly choose a plane to put line on */
        /*plane = ran_int(vars);
        do{
                y = ran_int(vars);
        }while(y == plane);*/
        plane = 0;
        y = 1;
        end1 = centre[plane] - 0.5;
        end2 = centre[plane] + 0.5;
        slope = 0.0;
        width = disp;
        tot = start;
        while(tot < start+samples)
        {
                /* choose a point on the line */
                x = (ran1()*(end2 - end1)) + end1;
                output = slope*x + yint; /* set y intercept*/
                data[tot][plane] = x;
                data[tot][y] = output + ran1()*width + (0.0 - (width/2.0));

                /* create tubular region around the line with radius = width/2*/
                for(v = 0; v < vars; v++)
                if((v != plane) && (v != y))
                        data[tot][v] = output + ran1()*width + (0.0 - (width/2.0));
                cat[tot] = clust_num;
                tot++;
        } /* end while samples */
}

void third_ring(int samples, int clust_num,double off_set,int start,
          int vars, double r,double width,double *centre)
{
        /* produces one third of a ring of distance r from centre */
        int tot,v,seed1,seed2;
        double x,y,output,angle,start_sec,sec_range;

        seed1 = seedx[clust_num];
        seed2 = seedy[clust_num];
        /* randomly choose one third of the unit circle */
        start_sec = 1.3333333333*ran1();
        sec_range = 0.66666666666;

        tot = start;
        while(tot < start+samples)
        {
                /* choose a point on the unit circle between
                start_sec and start_sec + 0.666666666*/
                angle = ((ran1()*sec_range) + start_sec)*PIE;

                /* scale in range of radius (r), place witin ring
                        and then shift orgin to coordinated given in centre array*/
                x = r*cos(angle) + /*width*gasdev()*/ ran1()*width + (0.0 - (width/2.0)) +
                                centre[seed1];
                y = r*sin(angle) + /*width*gasdev()*/ ran1()*width + (0.0 - (width/2.0)) +
                                centre[seed2];

                data[tot][seed1] = x;
                data[tot][seed2] = y;
                for(v = 0; v < vars; v++)
                        if((v != seed1) && (v != seed2))
                                data[tot][v] = r*sin(angle) + /*width*gasdev()*/ ran1()*width + (0.0 -
                                                        (width/2.0)) + centre[seed2];
                cat[tot] = clust_num;
                tot++;
        } /* end while samples */
} /*** end third_ring ***/

void t_third_ring(int samples, int clust_num,double off_set,int start,
            int vars, double r,double width,double *centre, double start_sec)
{
        /* produces 2/3 of a ring of distance r from centre */
        int tot,v,seed1,seed2;
        double x,y,output,angle,sec_range;

        /* randomly choose one third of the unit circle */
        seed1 = 0; /*seedx[clust_num]; */
        seed2 = 1; /*seedy[clust_num]; */
        sec_range = 1.333333333;
        tot = start;
        while(tot < start+samples)
        {
                /* choose a point on the unit circle between
                start_sec and start_sec + 0.666666666*/
                angle = ((ran1()*sec_range) + start_sec)*PIE;

                /* scale in range of radius (r), place witin ring
                        and then shift orgin to coordinated given in centre array*/
                x = r*cos(angle) + /*width*gasdev()*/ ran1()*width + (0.0 - (width/2.0)) +
                                centre[seed1];
                y = r*sin(angle) + /*width*gasdev()*/ ran1()*width + (0.0 - (width/2.0)) +
                                centre[seed2];
                data[tot][seed1] = x;
                data[tot][seed2] = y;
                for(v = 0; v < vars; v++)
                        if((v != seed1) && (v != seed2))
                                data[tot][v] = r*sin(angle) + /*width*gasdev()*/ ran1()*width + (0.0 -
                                                        (width/2.0))/* + centre[seed2]*/;
                /*      y + ran1()*width + (0.0 - (width/2.0));*/
                cat[tot] = clust_num;
                tot++;
        } /* end while samples */
} /*** end t_third_ring ***/

void ring(int samples, int clust_num,double off_set,int start,
          int vars, double r,double width,double *centre)
{
        /* produces one third of a ring of distance r from centre */
        int tot,v,seed1,seed2;
        double x,y,output,angle,start_sec,sec_range;

        seed1 = seedx[clust_num];
        seed2 = seedy[clust_num];
        tot = start;
        while(tot < start+samples)
        {
                /* choose a point on the unit circle */

                angle = ran1()*2.0*PIE;

                /* scale in range of radius (r), place witin ring
                and then shift orgin to coordinated given in centre array*/
                x = r*cos(angle) + width*gasdev() /* ran1()*width + (0.0 - (width/2.0))*/ +
                                centre[seed1];
                y = r*sin(angle) + width*gasdev() /* ran1()*width + (0.0 - (width/2.0))*/ +
                                centre[seed2];
                data[tot][seed1] = x;
                data[tot][seed2] = y;
                for(v = 0; v < vars; v++)
                        if((v != seed1) && (v != seed2))
                                data[tot][v] = r*sin(angle) + width*gasdev() /* ran1()*width + (0.0 -
                                                (width/2.0)) */ + centre[seed2];
                        /*      y + ran1()*width + (0.0 - (width/2.0));*/
                cat[tot] = clust_num;
                tot++;
        } /* end while samples */
} /*** end ring ***/

int ran_int(int num)
{ /* return uniform random deviate in range of 0 to num */

        int j;
        long k;
        static long iy = 0;
        static long iv[NTAB];
        double temp;
        if(*idum <= 0 || !iy)  /* intialize */
        {
                if(-(*idum) < 1)  /* prevent idum = 0; */
                        *idum = 1;
                else
                        *idum = -(*idum);
                for(j = NTAB+7;j >= 0 ; j--)   /* load shuffle table after 8 warm ups */
                {
                        k = (*idum)/IQ;
                        *idum = IA*(*idum - k*IQ) - IR*k;
                        if(*idum < 0)
                                *idum += IM;
                        if(j < NTAB)
                                iv[j] = *idum;
                }
        }

         k = (*idum)/IQ;
        *idum = IA*(*idum - k*IQ) - IR*k;
        if(*idum < 0)
        *idum += IM;
        j = iy/NDIV;
        iy = iv[j];
        iv[j] = *idum;
        if((temp = AM*iy) > RNMX)
                return((int)(RNMX*num));
        else
                return((int)(temp*num));
} /*** END RAN_INT ***/

int smooth(int num_pats, int num_clusts)
        /* ensures rounding errors do not prevent correct number of
        patterns being produced */

        int extra;

        if(num_pats%num_clusts == 0)
                return(num_pats/num_clusts);
        else
        {
                extra = num_pats%num_clusts;
                return((num_pats - extra)/num_clusts);
        }
}

void dist_matrix(int pats, int vars, int k)
{
        int p,c,v;

        for(p = 0; p < k - 1; p++)
        for(c = p + 1; c < k; c++)
        {
                d_matrix[p][c] = 0.0;
                for(v = 0; v < vars; v++)
```

```
                        d_matrix[p][c] += (centroids[p][v] - centroids[c][v])*
                                          (centroids[p][v] - centroids[c][v]);
                d_matrix[p][c] = d_matrix[p][c];
                d_matrix[c][p] = d_matrix[p][c];
        }
}

reset_pat_mems(int pats)
{
        int p;
        for(p = 0; p < pats;p++)
                mem[p] = c_mem[mem[p]];

}

void slink(int pats, int vars, int stop, int k)
{

        /* This procedure implements the single linkage clustering algorithm */

        int p,c,g,v,closest1,closest2,pc,pg,clusts;
        double min_dist,dist;

        for(c = 0;c < k; c++)
                c_mem[c] = c;
        if(DEBUG)
                printf("SLINK init clusts %d vars %d stop %d\n",k,vars,stop);
        clusts = k;

        /* calculate distance matrix */
        dist_matrix(pats,vars,k);
        while(clusts > stop) /* agglomerate until "stop" number of */
        {/* clusters have been found */
                min_dist = 1000000.0;
                closest1 = 0;
                closest2 = 0;
                for(c = 0; c < clusts - 1; c++)
                {
                        for(g = c + 1; g < clusts; g++) /* find two closest clusters */
                        {
                                /* for each two clusters, compare distance from all
                                c_members in one cluster, to all members of the other */
                                for(pc = 0; pc < k; pc++)
                                        for(pg = 0; pg < k; pg++)  /* find two closest obs.*/
                                                if((c_mem[pc] == c) && (c_mem[pg] == g))
                                                /* between clusts c and g */
                                                {
                                                        dist = d_matrix[pc][pg];
                                                        if(dist < min_dist)
                                                        {
                                                                closest1 = pc;
                                                                closest2 = pg;
                                                                min_dist = dist;
                                                        }
                                        }
                                } /* end g loop of find two closest clusters */
                        } /* end find two closest clusters routine */
                        c = c_mem[closest1];
                        g = c_mem[closest2];
                        for(p = 0; p < k; p++) /* re-index cluster memberships */
                        {
                                if(c_mem[p] == g)
                                        c_mem[p] = c;
                                if(c_mem[p] > g )
                                        c_mem[p]--;
                        }
                        clusts--;
                        if(DEBUG)
                        {
                                for(c = 0; c < k; c++)
                                        printf("%d ",c_mem[c]);
                                printf(" ::: clusts %d:  closest are %d
                                                %d\n",clusts,closest1,closest2);
                        }
                } /* end main agglomeration routine */
                /* update pattern memberships */
                for(p = 0; p < pats; p++)
                        mem[p] = c_mem[mem[p]];
                if(DEBUG)
                        printf("SLINK DONE\n");
} /* END SLINK */

void clink(int pats, int vars, int stop, int k)
{
        /* This procdure implements centroid linkage */

        int p,c,g,v,closest1,closest2,pc,pg,clusts,f;
        double min_dist,dist;
        int wins[MAXCLUSTERS],wf,wg,wfwg,skip[MAXCLUSTERS];

        for(c = 0; c < k; c++)
        {
                wins[c] = 0;
                skip[c] = 0;
        }
        for(p = 0; p < pats; p++)
                wins[ mem[p] ]++;
        for(c = 0;c < k; c++)
                c_mem[c] = c;

        if(DEBUG)
        printf("CLINK pats %d vars %d stop %d\n",pats,vars,stop);
        clusts = k;
        while(clusts > stop) /* agglomerate until "stop" number of */
        {               /* clusters have been found */
                min_dist = 1000000.0;
                closest1 = c;
                closest2 = g;
```

```
          for(c = 0; c < k - 1; c++)
                    for(g = c + 1; g < k; g++) /* find two closest clusters */
                    {
                              /* for each two clusters, compare distance between centroids */
                              if(!skip[c] && !skip[g])
                              {
                                        dist = 0.0;
                                        for(v = 0;v < vars;v++)
                                                  dist += (centroids[c][v] - centroids[g][v])*
                                                            (centroids[c][v] - centroids[g][v]);
                                                  if(dist < min_dist)
                                                  {
                                                            min_dist = dist;
                                                            closest1 = c;
                                                            closest2 = g;
                                                  }
                              }
                    } /* end find two closest clusters routine */
          g = closest1;
          f = closest2;
          wg = wins[g];
          wf = wins[f];
          if(wg == 0)
                    wg = 1;
          if(wf == 0)
                    wf = 1;
          wfwg = wins[f] + wins[g];
          if(wfwg == 0)
                    wfwg = 1;
          /*update centroids */
          for(v = 0; v < vars; v++)
                    centroids[g][v] = ((double)wg*centroids[g][v] +
                                                  (double)wf*centroids[f][v])/((double)(wfwg));
          wins[g] += wins[f];
          skip[closest2] = 1;
          if(DEBUG)
          {
                    for(c = 0; c < k; c++)
                    printf("%d ",c_mem[c]);
                    printf(" ::: clusts %d:  closest are %d
                                        %d\n",clusts,c_mem[closest1],c_mem[closest2]);
          }
          for(p = 0; p < k; p++) /* re-index cluster memberships */
          {
                    if(c_mem[p] == closest2)
                              c_mem[p] = closest1;
          }
          clusts--;
} /* end main agglomeration routine */
/* update pattern memberships */
for(p = 0; p < pats; p++)
          mem[p] = c_mem[mem[p]];
if(DEBUG)
          printf("CLINK DONE\n");
```

```
} /* END CLINK */

void alink(int pats, int vars, int stop, int k)
{ /* Average linkage */
        int p,c,g,v,closest1,closest2,pc,pg,clusts;
        double min_dist,dist,a_dist,pairs,skip[MAXCLUSTERS];

        for(c = 0;c < k; c++)
        {
                skip[c] = 0;
                c_mem[c] = c;
        }
        if(DEBUG)
                printf("AVERAGE pats %d vars %d stop %d\n",pats,vars,stop);
        clusts = k;
        /* calculate distance matrix */
        dist_matrix(pats,vars,k);
        while(clusts > stop) /* agglomerate until "stop" number of */
        {                       /* clusters have been found */
                min_dist = 1000000.0;
                closest1 = 0;
                closest2 = 0;
                for(c = 0; c < k - 1; c++)
                        for(g = c + 1; g < k; g++) /* find two closest clusters */
                        {
                                /* for each two clusters, compare distance from all
                                members in one cluster, to all members of the other */
                                if(!skip[c] && !skip[g])
                                {
                                        a_dist = 0.0;
                                        pairs = 0.0;
                                        for(pc = 0; pc < k - 1; pc++)
                                        for(pg = pc + 1; pg < k; pg++)
                                                if((c_mem[pc] == c) && (c_mem[pg] == g))
                                                {
                                                        a_dist += d_matrix[pc][pg];
                                                        pairs += 1.0;
                                                }
                                        a_dist = a_dist/pairs;
                                        if(a_dist < min_dist)
                                        {
                                                min_dist = a_dist;
                                                closest1 = c;
                                                closest2 = g;
                                        }
                        }
                } /* end find two closest clusters routine */
                skip[closest2] = 1;
                for(p = 0; p < k; p++) /* re-index cluster memberships */
                {
                        if(c_mem[p] == closest2)
                                c_mem[p] = closest1;
                }
                clusts--;
                if(DEBUG)
                {
                        for(c = 0; c < k; c++)
                                printf("%d ",c_mem[c]);
                        printf(" ::: clusts %d:  closest are %d %d\n",clusts,closest1,closest2);
                }
        } /* end main agglomeration routine */
        /* update pattern memberships */
        for(p = 0; p < pats; p++)
        mem[p] = c_mem[mem[p]];
        if(DEBUG)
                printf("ALINK DONE\n");
} /* END ALINK */

void ward(int pats, int vars, int stop, int k)
{
        /* Ward's method */

        int p,c,g,v,closest1,closest2,pc,pg,clusts;
        double min_dist,dist,*sum,winsc,winsg,sumg[MAXVARS];
        int wins[MAXCLUSTERS],wcwg,skip[MAXCLUSTERS];

        for(c = 0; c < k; c++)
        {
                wins[c] = 0;
                skip[c] = 0;
        }
        for(p = 0; p < pats; p++)
                wins[ mem[p] ]++;
        for(c = 0;c < k; c++)
        c_mem[c] = c;
        sum = (double*)malloc(vars*sizeof(double));
        if(DEBUG)
        printf("WARD pats %d vars %d stop %d\n",pats,vars,stop);
        clusts = k;
        while(clusts > stop) /* agglomerate until "stop" number of */
        {                    /* clusters have been found */
                min_dist = 10000000.0;
                closest1 = 0;
                closest2 = 0;
                for(c = 0; c < k - 1; c++)
                        for(g = c + 1; g < k; g++)
                        {
                        if(!skip[c] && !skip[g])
                        {
                                for(v = 0; v < vars; v++)
                                sum[v] = sumg[v] = 0.0;
                                winsc = winsg = 0.0;
                                for(p = 0; p < k;p++)
                                {
                                        if(c_mem[p] == c)
                                        {
                                                for(v = 0; v < vars; v++)
                                                sum[v] += centroids[p][v];
                                        winsc += 1.0;
                                }
                                if(c_mem[p] == g)
                                {
                                        for(v = 0; v < vars; v++)
                                        sumg[v] += centroids[p][v];
                                        winsg += 1.0;
                                }
                        }
                        for(v = 0; v < vars;v++)
                        {
                                sum[v] /= winsc;
                                sumg[v] /= winsg;
                        }
                        dist = 0.0;
                        for(v = 0; v < vars;v++)
                                dist += (sum[v] - sumg[v])*(sum[v] - sumg[v]);
                        dist = dist/((1.0/winsc) + (1.0/winsg)) ;
                        if(dist < min_dist)
                        {
                                min_dist = dist;
                                closest1 = c;
                                closest2 = g;
                        }
                }
        } /* end find two closest clusters routine ( C & G LOOP )*/
        skip[closest2] = 1;
        for(p = 0; p < k; p++) /* re-index cluster memberships */
                if(c_mem[p] == closest2)
                        c_mem[p] = closest1;
        wins[closest1] += wins[closest2];
        clusts--;
        if(DEBUG)
        {
                for(c = 0; c < k; c++)
                        printf("%d ",c_mem[c]);
                printf(" ::: clusts %d:  closest are %d %d\n",clusts,closest1,closest2);
        }
    } /* end main agglomeration routine */
    /* update pattern memberships */
    for(p = 0; p < pats; p++)
    mem[p] = c_mem[mem[p]];
    if(DEBUG)
        printf("WARD DONE\n");
    free(sum);
} /* END ward */

int wong(int pats, int vars, int stop, int k)
{
    /* Wong's hybrid method */

    double wss[MAXCLUSTERS];
    double midpt[MAXVARS];
    double d_g,d_c,dist,min_dist,numer,denom;
```

```
int p,c,v,g,flagg,flagc,x,clusts,closest1,closest2,pc,pg;
int wins[MAXCLUSTERS],no_more_flag;

/* get wins and wss */
for(c = 0; c < k; c++)
{
          wins[c] = 0;
          wss[c] = 0.0;
          c_mem[c] = c;
          for(g = 0; g < k;g++)
          d_matrix[c][g] = 0.0;
}

for(p = 0; p < pats; p++)
{
          wins[mem[p]]++;
          for(v = 0;v < vars; v++)
          wss[mem[p]] += (centroids[mem[p]][v] - data[p][v])*
                              (centroids[mem[p]][v] - data[p][v]);
}
/* find adjacent clusters */
for(c = 0; c < k; c++)
          for(g = 0; g < k; g++)
          {
                    if(g == c)
                    {
                              d_matrix[c][g] = -9.0;
                              continue;
                    }
                    /* calc midpoint */
                    for(v = 0; v < vars;v++)
                              midpt[v] = (centroids[c][v] + centroids[g][v])/2.0;
                    /* find dist of midpoint to all other clusters */
                    for(v = 0,d_g = 0.0,d_c = 0.0; v < vars;v++)
                    {
                              d_g += (midpt[v] - centroids[g][v])*(midpt[v] - centroids[g][v]);
                              d_c += (midpt[v] - centroids[c][v])*(midpt[v] - centroids[c][v]);
                    }
                    d_g = sqrt(d_g);
                    d_c = sqrt(d_c);
                    flagg = 1;
                    flagc = 1;
                    for(x = 0; x < k;x++)
                    {
                              if((x != c) && (x != g))
                              {
                                        dist = 0.0;
                                        for(v = 0; v < vars; v++)
                                                  dist += (midpt[v] - centroids[x][v])*(midpt[v] -
                                                                      centroids[x][v]);
                                        dist = sqrt(dist);
                                        if(dist < d_g)
                                                  flagg = 0;
                                        if(dist < d_c)
```

```
                        flagc = 0;
                    }
                }
                /* if mid point is closer to g or c than any other*/
                    /* calculate wong distance (density at midpoint of c and g */

                    if(flagg || flagc)
                    {
                        for(v = 0,dist = 0.0;v < vars;v++)
                        dist += (centroids[g][v] - centroids[c][v])*(centroids[g][v] -
                                centroids[c][v]);
                        numer = pow((wss[g] + wss[c])  + 0.25*dist*((double)(wins[c] +
                                wins[g])),(double)vars/2.0);
                        denom = pow((double)(wins[c] + wins[g]),1.0 + (double)vars/2.0);
                        d_matrix[c][g] = (numer/denom)*100000.0;
                    }
                    else
                        d_matrix[c][g] = -9.0;
            }
            /* using the density based distance matrix, run single linkage */
            clusts = k;
            while(clusts > stop)        /* agglomerate until "stop" number of */
            {                   /* clusters have been found */
                min_dist = 1000000.0;
                closest1 = 0;
                closest2 = 0;
                no_more_flag = 0;
                for(c = 0; c < clusts - 1; c++)
                {
                    for(g = c + 1; g < clusts; g++) /* find two closest clusters */
                    {
                        /* for each two clusters, compare distance from all
                        c_members in one cluster, to all members of the other */
                        for(pc = 0; pc < k; pc++)
                            for(pg = 0; pg < k; pg++)  /* find two closest obs.*/
                                if((c_mem[pc] == c) && (c_mem[pg] == g) &&
                                        (d_matrix[pc][pg]   >   -1.0))    /*
    between clusts                          c and g */
                                {
                                    no_more_flag = 1;
                                    dist = d_matrix[pc][pg];
                                    if(dist < min_dist)
                                    {
                                        closest1 = pc;
                                        closest2 = pg;
                                        min_dist = dist;
                                    }
                                }
                    }  /* end g loop of find two closest clusters */
                } /* end find two closest clusters routine */
                if(!no_more_flag)
                    return(0);
                c = c_mem[closest1];
                g = c_mem[closest2];
```

```
			for(p = 0; p < k; p++) /* re-index cluster memberships */
			{
				if(c_mem[p] == g)
					c_mem[p] = c;
				if(c_mem[p] > g)
					c_mem[p]--;
			}
			clusts--;
			/* printf("WONG %d\n",clusts);*/
			if(DEBUG)
			{
				for(c = 0; c < k; c++)
					printf("%d ",c_mem[c]);
				printf("::: %d %d %lf\n",closest1,closest2,d_matrix[closest1][closest2]);
			}
		} /* end main agglomeration routine */
		/* update pattern memberships */
		for(p = 0; p < pats; p++)
			mem[p] = c_mem[mem[p]];
		if(DEBUG)
		printf("WONG DONE\n");
		return(1);
} /* END WONG */

int get_partition(int pats,int vars,int num_clusts)
{
		int c,g,p,maxc,minc,v,largest,flag,q;
		int wins[MAXCLUSTERS],change,acts[MAXCLUSTERS][MAXCLUSTERS];
		double dmax,dmin,min_cent[MAXVARS],max_cent[MAXVARS],dist;
		double sd[MAXVARS];

		/* This procedure is the pattern partitioning stage of both APC and WONG'S method.
			Basically, it drives the moving method. If a cluster is found with less than
			MINWINS patterns, the cluster is deleted and the moving method is rerun with
			the remaining centroids as intial seeds and the number of clusters decremeted by 1. The
			final number of clusters is returned. */

		seed(pats,num_clusts,vars);
		flag = 1;
		while(flag) /* remove spurious clusters */
		{
			flag = 0;
			moving(pats,vars,num_clusts,0);/* run pattern partitioning algo */
			get_mem(num_clusts,vars,pats);
			for(c = 0; c < num_clusts; c++)
				wins[c] = 0;		/* get number of wins per cluster */
			for(p = 0; p < pats; p++)
				wins[ mem[p] ]++;
			for(c = 0; c < num_clusts;c++) /* remove spurioius clusters */
			{
				if(wins[c] <= MINWINS)
				{
					for(q = c;q < num_clusts - 1;q++)
					{
```

```
                                        wins[q] = wins[q + 1];
                                        for(v = 0; v < vars; v++)
                                                centroids[q][v] = centroids[q + 1][v];
                                }
                                num_clusts--;
                                flag = 1;
                        }
                } /* end remove spurious clusters */
        }
        return(num_clusts);
}/* end partitions */

/***** APC *******/

int apc(int k, int vars, int pats, int stop)
{
        int c,clusts,g,p,v,closest1,closest2,pg,pc;
        double sd,dist,min_dist,last,*widths;
        int wins[MAXCLUSTERS];
        int c_mem[MAXCLUSTERS];

        widths = (double *)malloc(MAXCLUSTERS*sizeof(double));
        for(c = 0; c < k;c++)
        {
                c_mem[c] = c;
                widths[c] = 0.0;
                wins[c] = 0;
        }
        /* get cluster widths */
        for(p = 0; p < pats;p++)
        {
                for(v = 0; v < vars;v++)
                widths[mem[p]] += (centroids[mem[p]][v] - data[p][v])*
                                (centroids[mem[p]][v] - data[p][v]);
                wins[mem[p]]++;
        }
        for(c = 0; c < k;c++)
        {
                if(wins[c] < 1)
                        wins[c] = 1;
                widths[c] = sqrt(widths[c]/(double)wins[c]);
        }
        for(c = 0; c < k; c++)
                for(g = 0;g < k;g++)
                        d_matrix[c][g] = 1.0;
        adj(pats, vars, stop, k);
        if(DEBUG)printf("GOT WIDTHS\n");
        for(c = 0; c < k - 1; c++)  /*for each cluster find intercluster density between*/
                for(g = c + 1;g < k;g++)
                {
                        if(d_matrix[c][g] > -1.0)
                        {
                                if(DEBUG)printf("Evaluating clust: %d and clust: %d\n",c,k);
                                sd = get_sd(widths[c],widths[g]);
                                d_matrix[c][g] = check_density(g,c,vars,pats,SEG_WIDTH*sd);
                        }
                }
        clusts = k;
        last = 9999999.0;
        while(clusts > stop) /* agglomerate until stop clusters are found */
        {
                min_dist = -99999999.0;
                closest1 = -1;
                closest2 = -1;
                for(c = 0; c < k - 1; c++)
                        for(g = c + 1; g < k; g++)
                        {
                                dist = d_matrix[c][g];
                                if((dist > -1.0) && (dist > min_dist)&& (c_mem[c] != c_mem[g]))
                                {
                                        min_dist = dist;
                                        closest1 = c;
                                        closest2 = g;
                                }
                        }/* end find closest clusters */
                if((min_dist < 0.0) || (closest1 == -1)) /* no more distances > 0, */
                        break                   /* nothing else can be agglomerated */
                c = c_mem[closest1];
                g = c_mem[closest2];
                for(p = 0; p < k; p++) /* re-index cluster memberships */
                if(c_mem[p] == g)
                        c_mem[p] = c;
                d_matrix[closest1][closest2] = -1.0;
                d_matrix[closest2][closest1] = -1.0;
                clusts--;
        }/* end cluster agglomeration */
        /* update pattern memberships */
        for(p = 0; p < pats; p++)
                mem[p] = c_mem[mem[p]];
        if(DEBUG) printf("\n\nAPC DONE\n");
        free(widths);
} /* END APC */

double check_density(int clust2,int clust1,int vars,int pats, double width)
{
/* this procedure returns the density based distance between cluster centroids in APC */

        double B,bin_size,dist,d;
        int densities[MAXSEGMENTS];
        int p,c;
        int v,num_bins,q,max_den,min_den;
        int num_pats;
        double volume;

        if(DEBUG)printf("Checking density between %d and %d\n",clust1, clust2);
        B = 0.0; /* get lenght of potential line segemnt */
        for(v = 0; v < vars; v++)
                B += (centroids[clust1][v] - centroids[clust2][v])*
```

```
                    (centroids[clust1][v] - centroids[clust2][v]);
    B = sqrt(B);
    num_pats = 0;
    for(p = 0;p < pats; p++)
    {       /* count number of patterns between centroids */
            /* this is used for calculating the bin size */
            dist = near_line_seg(B*B,p,clust1,clust2,vars,width);
            if(dist >= 0.0)
                    num_pats++;
    }
    if(num_pats < 10)
            bin_size = B/3.0;
    else
            bin_size = (B/floor(sqrt((double)num_pats)));
    num_bins = 0;
    for(d = 0,q = 0; d <= B; d += bin_size,q++)
    {
            num_bins++;
            densities[q] = 0;
    }
    /* find all points within 1 sd of the connnecting line */

    for(p = 0;p < pats; p++)
    {
            dist = near_line_seg(B*B,p,clust1,clust2,vars,width);
            if(dist >= 0.0)
            {
                    for(d = 0.0,q = 0; d <= B - bin_size; d += bin_size,q++)
                            if((dist > d) && (dist <= d + bin_size))
                    densities[q]++;
            }
    }
    max_den = 0;
    min_den = 32000;
    for(d = 0.0,q = 0; d <= B - bin_size;d += bin_size,q++)
    {
            if(densities[q] > max_den)
                    max_den = densities[q];
            if(densities[q] < min_den)
                    min_den = densities[q];
    }
    if(max_den == 0)
            return(0.0);
    if(ADD == 1)
            return((double)min_den);
    else if(ADD == 0)
            return((((double)min_den)/((double)max_den)));
    else
    {
            volume = PIE*width*width*bin_size;
            return(((double)min_den)/volume);
    }
} /* END CHECK_DENSITY */
```

```
double near_line_seg(double B, int pat,int clust1, int clust2,int vars,
                    double width)
{
        /* procedure determines if a pattern is inside the hypercylinder
                determioned by the centrods of clust1 and clust2. If the pattern is indise (or on) the
                        hypercyllinder, the distnce of the pattern to the axis of the hypercykener is
returned.                               Ohterwise, -1 is returned */

        int v;
        double dist;
        double A,C,Z;

        A = C = 0.0;
        for(v = 0; v < vars; v++)
        {
                A += (centroids[clust1][v] - data[pat][v])*
                        (centroids[clust1][v] - data[pat][v]);
                C += (centroids[clust2][v] - data[pat][v])*
                        (centroids[clust2][v] - data[pat][v]);
        }
        if(A == 0.0)
                return(0.0); /* pattern and centroid are the same */
        if(C == 0.0)
                return(sqrt(B));
        if((B > 0.0)&&(fabs((A - C)/B) <= 1.0))
        {
                Z = ((A + B - C)*(A + B - C))/(4.0*B);
                dist = A-Z;
                if(dist < 0.0)
                {
                        printf("DIST IS GREATER THAN A, a^2 - dist^2 = %lf\n",(A) - (Z));
                        printf("A %f, B %f, C %f Z %f dist %f\n",A,B,C,Z,dist);
                        dist = 0.0;
                        printf("c1   c2   pat\n");
                        for(v = 0; v < vars;v++)
                                printf("%.4lf %.4lf  %.4lf\n",centroids[clust1][v],
                                                centroids[clust2][v],data[pat][v]);
                        getchar();
                }
                if(sqrt(dist) <= width)
                        return(sqrt(A - dist));
        }
         return(-1.0);
}/ * END NEAR_LINE_SEG */

double get_sd(double wc,double wg)
{
        if(SD_TYPE == 0) /* return smallest of the two sd's */
        {
                if(wc < wg)
                        return(wc);
                else
                        return(wg);
        }
```

```
                else if(SD_TYPE == 1) /* retrun largest of the two sd's */
                {
                        if(wc > wg)
                                return(wc);
                        else
                                return(wg);
                }
        else /* return average of the two sd'd */
                return((wc + wg)/2.0);
}

int adj(int pats, int vars, int stop, int k)
{
        /* this procedure finds adjacent cluster centroids. Non-adjacent centroids have -2.0 entered into
                        the corresponding elements of the distance matrix */

        double wss[MAXCLUSTERS];
        double midpt[MAXVARS];
        double d_g,d_c,dist,min_dist;
        int p,c,v,g,flagg,flagc,x,clusts,closest1,closest2,pc,pg;
        int wins[MAXCLUSTERS],no_more_flag;

        /* find adjacent clusters */
        for(c = 0; c < k - 1; c++)
                for(g = c + 1; g < k; g++)
                {
                        /* calc midpoint */
                        for(v = 0; v < vars;v++)
                                midpt[v] = (centroids[c][v] + centroids[g][v])/2.0;
                        /* find dist of midpoint from both clusters */
                        for(v = 0,d_g = 0.0,d_c = 0.0; v < vars;v++)
                        {
                                d_g += (midpt[v] - centroids[g][v])*(midpt[v] - centroids[g][v]);
                                d_c += (midpt[v] - centroids[c][v])*(midpt[v] - centroids[c][v]);
                        }
                        d_g = sqrt(d_g);
                        d_c = sqrt(d_c);
                        flagg = 1;
                        flagc = 1;
                        for(x = 0; x < k;x++)
                        {
                                if((x != c) && (x != g))
                                {
                                        dist = 0.0;
                                        for(v = 0; v < vars; v++)
                                                dist += (midpt[v] - centroids[x][v])*(midpt[v] -
                                                                centroids[x][v]);
                                        dist = sqrt(dist);
                                        if(dist < d_g)
                                flagg = 0;
                                if(dist < d_c)
                                        flagc = 0;
                        }
                }
                /* if mid point is closer to g or c than any other*/
                /* calculate wong distance (density at midpoint of c and g */
                if(!flagg && !flagc)
                {
                        d_matrix[c][g] = -2.0;
                        d_matrix[g][c] = d_matrix[c][g];
                }
        }
        return(1);
}
```

References

Adriaans, P. and Zantinge, D. (1996) *Data Mining*. Addison-Wesly, Harlow, England.

Al-Sultan, K. S. (1995) A tabu search approach to the clustering problem. *Pattern Recognition*, 28 (9), 1443-1451.

Anderberg, M. R. (1973) *Cluster analysis for Applications*. Academic Press, London.

Arabie P. and Hubert, L. J. (1966) An overview of combinatorial data analysis. In P. Arabie, J. Hubert and G. De Soete (eds). *Clustering and Classification*. World Scientific, River Edge, N. J., 5 - 63.

Astrahan, M. M. (1970) Speech analysis by clustering, or the hyperphoneme method. *Stanford Artificial Intelligence Proj. Mem.*, AIM-124, AD 709067, Stanford Univ. Stanford, California.

Atlas R. S. and Overall, J. E. (1994) Comparative evaluation of two superior stopping rules for hierarchical cluster analysis. *Psychometrika*, 59(4), 581-591.

Babu, G. P. and Murty, M. N. (1994) Clustering with evolutionary strategies. *Pattern Recognition*, 27(2), 321 - 329.

Babu, G. P. and Murty, M. N. (1993) A near optimal seed value selection in k-means algorithm using a genetic algorithm. *Pattern Recognition Letters*, 14, 763 - 69.

Backer, E. (1995) *Computer assisted Reasoning in Cluster analysis*. Prentice Hall, London.

Balakrishnan, P. V. Cooper, M. C., Jacob, V. S. and Lewis, P. A (1996) Comparative performance of the FSCL neural network and k-means for market segmentation. *European Journal of Operational Research*, 93(2), 346 - 57.

Balakrishnan, P. V. Cooper, M. C., Jacob, V. S. and Lewis, P.A. (1994) A study of the classification capabilities of neural networks using unsupervised learning: A comparison with k-means clustering. *Psychometrika*, 59(4), 509 - 24.

Balasubramaniam, A. Parthasarathy, G. and Chatterji, B. N. (1990) Knowledge based approach to cluster analysis selection. *Pattern Recognition Letters*, 11, 651 - 61.

Ball, G. H. and Hall, D. J. (1967) *PROMENADE - an on-line pattern recognition system.* Rep. No. RADC-TR-67-310, AD 822174. Stanford Res. Inst., Menlo Park, California.

Ball, G. H. and Hall, D. J. (1965) *ISODATA, A novel method of data analysis and pattern classification.* AD 699616, Stanford Res. Inst., Menlo Park, Calif.

Bayne, C. K. , Beauchamp, J. J., Begovich, C. L. and Kane, V. E. (1980) Monte Carlo comparisons of selected clustering procedures. *Pattern Recognition*, 12, 51 - 62.

Beale, E. M. L. (1969) Euclidean cluster analysis, *Bulletin of the International Statistical Institute,* 43, 92 - 4.

Bezdek, J. C. (1981) *Pattern Recognition with Fuzzy Objective Functions.* Plenum Press, New York.

Bishop C. M., Svensén M. and Williams C. K. (1997) GTM: a principled alternative to the self-organizing map. In *Advances in Neural Information Processing Systems 9*, MIT press.

Blashfield, R. K. (1977) *A consumer report on cluster analysis software: (3) iterative partitioning methods (NSF grant DCR 74-20007)*, State College, PA, Pennsylvania State University, Department of Psychology.

Blashfield, R. K. (1976) Mixture model tests of cluster analysis: accuracy of four agglomerative hierarchical methods. *Psychological Bulletin*, 83(3), 377 - 88.

Blashfield, R. K. and Morey L. C. (1980) A comparison of four clustering methods using MMPI Monte Carlo data. *Applied Psychological Measurement*,

Bonner, R. E. (1964) On some clustering techniques. *IBM J. Res. Dev.*, 8, 22 - 32.

Brunelli, R. (1992) Optimal histogram partitioning using a simulated annealing technique. *Pattern Recognition Letters*, 13, 581 - 586.

Carmichael, J. W., George J. A. and Julius, R. S. (1968) Finding natural clusters. *Syst. Zool.*, 17, 144 - 50.

Carpenter, G. A. and Grossberg, S. (1987) ART2: self organization of stable category recognition codes for analog input patterns. *Applied Optics*, 26, 4919 - 4930.

Carpenter, G. A., Grossberg, S. and Rosen, D. B. (1991) Fuzzy Art: fast stable learning and categorization for analog patterns by an adaptive resonance system. *Neural Networks*, 4(6), 759-771.

Cattell R. B. and Coulter, M. A. (1966) Principals of behavioural taxonomy and the mathematical basis of the taxonome computer program. *British Journal of Mathematical and Statistical Psychology*, 19, 237 - 69.

Celeux, G. and Govaert, G. (1995) Gaussian parsimonious clustering models. *Pattern Recognition*, 28(5), 781 - 93.

Chaudhuri, D. and Chaudhuri B. B. (1997) A novel multiseed nonhierarchical data clustering technique. *IEEE Transactions on Systems, Man and Cybernetics-Part B: Cybernetics*, 27(5), October, 871 - 877.

Chaudhuri, D. and Chaudhuri B. B. (1995) A multiseed non-hierarchical clustering technique for data analysis. *International Journal of Systems Science*, 25(2), 375 - 385.

Chaudhuri, D., Chaudhuri, B. B. and Murthy, C. A. (1992) A new split and merge clustering technique. *Pattern Recognition Letters*, 13, 399 - 409.

Chen, S. K., Mangiameli, P. and West, D. (1995) The comparative ability of self-organizing neural networks to define cluster structure. *Omega International Journal of Management Science*, 23(3), 271 - 9.

Chen B. and Titteringdon, D. M. (1994) Neural networks: A review from a statistical perspective. *Statistical Science*; 9(1), 2-54.

Cheng R. and Milligan (1996) G. W. K-means clustering with influence detection. *Educational and Psychological Measurement*, 56(5), 833 - 838.

Chinrungrueng C. and C. H. Sèquin (1995) Optimal adaptive k-means algorithm with dynamic adjustment of learning rate. *IEEE Transactions on Neural Networks*, 6(1), 157 - 169.

Choi, Doo-Il and Park, Sang-Hui (1994) Self-creating and organizing neural networks. *IEEE Transactions on Neural Networks*, 5(4), 561 - 575.

Cormack, R. M. (1971) A review of classification (with Discussion). *Journal of the Royal Statistical Society,* Series *A*, 134, 321 - 367.

Cowgill, M. C. (1993) *Monte Carlo Validation of Two Genetic Clustering Algorithms*. Ph.D. Thesis, Virginia Polytechnic Institute and State University.

Dong, D. and McAvoy T. J. (1996) Nonlinear principal component analysis based on principal curves and neural networks. *Computers and Chemical Engineering* 20(1), 65 - 78.

Doyle, L. B. (1966) *Breaking the cost barrier in automatic classification*. Prof. Pap. SP-2516, AD 636837. Systems Develop. Corp., Santa Monica, California.

Dubes, R. and Jain, A. K. (1976) Clustering techniques: the user's dilemma. *Pattern Recognition*, 8, 247 - 60.

Duda, R. O. and Hart, P. E. (1973) *Pattern Classification and Scene Analysis*. Wiley, New York.

Edelbrock, C. (1979) Mixture model tests of hierarchical clustering algorithms: the problem of classifying everybody. *Multivariate Behavioural Research*, 14, 367 - 84.

Evans, I. S. (1983) Bivariate and multivariate analysis: relationships between variables. In D. W. Rhind (ed) *A Census User's Handbook*, Methuen, London.

Everitt, B. S. (1993) *Cluster Analysis*. 3rd Ed., Heinemann Educational, London.

Everitt, B. S. (1981) *Cluster Analysis*. 2nd Ed., Heinemann Educational, London.

Fielding, A (1977) Binary segmentation. The Automatic Interaction Detector and related techniques for exploring data structures. In *Analysis of survey data*, Vol. I, Eds. C.A. O'Muircheartaigh and C. Payne. Wiley, New York.

Fisher L. and Van Ness (1971) Admissible clustering procedures. *Biometrika*, 58(91).

Forgy, E. W. (1965) Cluster analysis of multivariate data: Efficiency versus interpretability of classifications. Biometrics Society Meetings, Riverside, Calif. (Abstract in *Biometrics*, 21(3), 768).

Frigui, H. and Krishnapuram, R. (1997) Clustering by competitive agglomeration. *Pattern Recognition*, 30(7), 1109 - 1119.

Fritzke, B. (1995) Growing grid - a self-organizing network with constant neighbourhood range and adaptation strength. *Neural Processing Letters*; 2(5), 9 - 13.

Fritzke, B. (1991) Unsupervised clustering with growing cell structures. In *Proc. Int. Joint Conf. on Neural Networks*, Seattle, WA, Vol. 2, 531 - 536.

Fromm, F. R. and Northouse, R. A. (1976) Class: A nonparametric clustering algorithm. *Pattern Recognition*, 8, 107 - 14.

Gengerelli, J. A. (1963) A method for detecting subgroups in a population and specifying their membership. *Journal of Psychology*, 5, 456 - 468.

Gordon A. D. (1981) *Classification: Methods for the Exploratory Analysis of Multivariate Data*. Chapman and Hall, London.

Gower, J. C. (1967) A comparison of some methods of cluster analysis, *Biometrics*, 23, 623 - 37.

Gross, A. L. A. (1972) A Monte Carlo study of the accuracy of a hierarchical grouping procedure. *Multivariate Behavioral Research*, 7, 379 - 89.

Gupta, L. and Tammana, R. (1995) A discrepancy measure for improved clustering. *Pattern Recognition*, 28(10), 1627-1634.

Hand, D. J. (1981) *Discrimination and Classification*. John Wiley and Sons, New York.

Hartigan, J. A. (1975) *Clustering Algorithms*. Wiley, New York.

Haykin, S. (1994) *Neural Networks: A Comprehensive Foundation*. Maxwell Macmillan Int., New York.

Helmstadter G. C. (1957) An empirical comparison of methods for estimating profile similarity. *Educational and Psychological Measurement*, 17, 71 - 82.

Henrichen E. G. and Fu, K. S. (1968) On mode estimation in pattern recognition. *Proc. 7th Symp. Adaptive Process.*, UCLA.

Hathaway, R. J. and Bezdek, C. (1995) Optimization of clustering criteria by reformulation. *IEEE Transactions on Fuzzy Systems*, 3(2), 241 - 245.

Hertz, J., Krogh A. and Palmer R. G. (1991) *Introduction to the theory of neural networks*. Addison-Wesley, Redwood City, Calif.,.

Horn, D. (1943) A study of personality syndromes. *Character and Personality*, 12, 257-274.

Hubert, L. and Arabie, P. (1985) Comparing partitions. *Journal of Classification*, 2, 193 - 218.

Ismail, M. A. and Kamel, M. S. (1989) Multi-dimensional data clustering utilizing hybrid search strategies. *Pattern Recognition*, 22, 75 - 89.

Jain, A. K. and Dubes, R. C. (1988) *Algorithms for Clustering Data*. Prentice Hall, Englewood Cliffs, NJ.

Jardine, N. and Sibson, R. (1971) *Mathematical Taxonomy*. Wiley, London.

Jawahar, C.V., Biswas, P. K. and Ray A. K (1995) Detection of clusters of distinct geometry: a step towards generalized fuzzy clustering. *Pattern Recognition Letters*, 16, 1119 - 1123.

Johnson, S. C. (1967) Hierarchical clustering schemes. *Psychometrika*, 32(3),241 - 254.

Kangas, J. A., Kohonen, T. K. and Laaksonen, J. T. (1990) Variants of self-organizing maps. *IEEE Transactions on Neural Networks*; 1(1), 93-99.

Kant, S., Rao, T. L. and Sundaram, P. N. (1994) An automatic and stable clustering algorithm. *Pattern Recognition Letters*, 15, 543-549.

Kendall, M. G. and Buckland, W. R. (1982) *A Dictionary of Statistical Terms, 4th ed.* Longman, London.

Klein, R. W. and Dubes R. C. (1989) Experiments in projection and clustering by simulated annealing. *Pattern Recognition*, 22(2), 213-220.

Kohonen, T. (1995) *Self-Organization and Associative Memory*, 3rd ed., Springer - Verlag.

Kohonen, T. (1982) Self-organized formation of topologically correct feature maps, *Biological Cybernetics*, 43, 59 - 69.

Koikkalainen, P. (1995) Fast deterministic self-organizing maps. In *Proceedings of ICANN'95, International Conference on Artificial Neural Networks*, Vol. II, EC2 & Cie, Paris, 63-68.

Koikkalainen, P. (1994) Progress with the tree-structured self-organizing map. In *Proceedings of ECAI'94, 11th European Conference on Artificial Intelligence*, Wiley, Chichester, England, 211-215 .

Krzanowski, W. J. (1993) *Principles of Multivariate Analysis: A User's Perspective.*, Oxford University Press, Oxford.

Kruskal, J. B. Jr. (1956) On the shortest spanning subtree of a graph and the travelling salesman problem. *Proceedings of the American Mathematical Society*, No. 7, 48-50.

Kuiper, F. K. and Fisher, L. (1975) 391: A Monte Carlo comparison of six clustering procedures. *Biometrics*, 31, 777 - 83.

Lambert, J. M and Williams, W. T. (1966) Multivariate methods in plant ecology IV: Comparison of information analysis and association analysis. *Journal of Ecology*, 54, 635 - 64.

Lance, G. N. and Williams, W. T. (1967a) A general theory of classificatory sorting strategies I. Hierarchical systems. *The Computer Journal*, 9, 373-380.

Lance, G. N. and Williams, W. T. (1967b) A general theory of classificatory sorting strategies II. Clustering systems. *Computer Journal*, 10(3), 271-76.

Liu, S-T, and Tsai, W-H. (1989) Moment-preserving clustering. *Pattern Recognition*, 22(4), 433 - 47.

Lorr, M. (1983) *Cluster Analysis for Social Scientists*. Jossey-Bass, San Francisco.

MacNaughton-Smith, P, Williams, W. T., Dale, M. B. and Mockett, L. G. (1964) Dissimilarity analysis, *Nature*, 202, 1034 - 35.

MacQueen, J. B. (1967) Some methods of analysis and classification of multivariate observations. *Proc. Symp. Math. Statist. and Probability*, 5th, Berkeley, 1, 281-97.

Mangiameli, P., and Chen, S. K. and West, D. (1996) A comparison of SOM neural network and hierarchical clustering methods. *European Journal of Operational Research*; 92, 402 - 417.

Massart, D. L., Plastria, F. and Kaufman, L. (1983) Non-hierarchical clustering with MASLOC. *Pattern Recognition*, 16(5), 507 - 15.

McQuitty, L. L. (1957) Elementary linkage analysis for isolating orthogonal and oblique types of typal relevancies. *Educational and Psychological Measurement*, 17, 297 - 29.

McRae, D. J. (1971) Mika: a FORTRAN IV iterative k-means cluster analysis program. *Behavioral Science*, 16(4), 423-24.

Merz, C. J., & Murphy, P.M. (1998). *UCI Repository of machine learning databases* [http://www.ics.uci.edu/~mlearn/MLRepository.html]. Irvine, CA: University of California, Department of Information and Computer Science.

Messatfa, H. (1992) An algorithm to maximize agreement between partitions. *Journal of Classification*, 9, 5-15.

Mezzich, J. (1978) Evaluating clustering methods for psychiatric-diagnosis. *Biological Psychiatry*, 13, 265-81.

Milligan G. W. (1996) Clustering validation: results and implications for applied analyses. In P. Arabie, J. Hubert and G. Se Soete (eds). *Clustering and Classification*. World Scientific, River Edge, N. J., 341 -375.

Milligan G. W. (1985) An algorithm for generating artificial test clusters. *Psychometrika*. 50(1), 123 - 127.

Milligan, G. W. (1981a) A review of Monte Carlo tests of cluster analysis. *Multivariate Behavioral Research*, 16, 379 - 407.

Milligan, G. W. (1981b) A Monte Carlo study of thirty internal criterion measures for cluster analysis. *Psychometrika*, 46(2), 187 - 199.

Milligan, G. W. (1980) An examination of the effect of six types of error perturbation on fifteen clustering algorithms. *Psychometrika*, 45(3), 325 - 42.

Milligan G. W. and Cooper, M. C. (1987) Methodology review: Clustering methods. *Applied Psychological Measurement*, 11(4), 329 - 354.

Milligan G. W. and Cooper M. C. (1986) A study of the comparability of external criteria for hierarchical cluster analysis. *Multivariate Behavioral Research*, 21, 441 - 58.

Milligan G. W. and Cooper, M. C. (1985) An examination of procedures for determining the number of clusters in a data set. *Psychometrika* 50(2), 159 - 179.

Milligan, G. W. and Isaac, P. D. (1980) The validation of four ultrametric clustering algorithms. *Pattern Recognition*, 12, 41 - 50.

Milligan, G. W., Soon, S. C. and Sokol, L. M. (1983) The effect of cluster size, dimensionality and the number of clusters on recovery of true cluster structure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-5(1), 40 - 47.

Mojena R. (1977) Hierarchical grouping methods and stopping rules: an evaluation. *The Computer Journal*, 20(4), 359-63.

Moody, J. and Darken, C. J. (1989) Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1(2), 281 - 294.

Murtagh, F. (1995) Interpreting the Kohonen self-organizing feature map using contiguity-constrained clustering. *Pattern Recognition Letters*, 16, 399 - 408.

Murtagh, F. (1992) Comments on "Parallel algorithms for hierarchical clustering and validity." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(10), 1056 - 1057.

Murtagh, F. (1983) A survey of recent advances in hierarchical clustering algorithms. *The Computer Journal*, 26(4), 354 - 359.

Murtagh, F. and Hernández-Pajares, M. (1995) The Kohonen self-organising map method: An assessment. *Journal of Classification*; 12, 165-90.

Nour M. A. and Madey, G. R. (1996) Heuristic and optimization approaches to extending the Kohonen self organizing algorithm. *European Journal of Operational Research*, 93, 428 - 448.

Openshaw, S. (1995) *Census User's Handbook*, Pearson Professional Ltd., London

Osbourn, G. C. and Martinez, R. F. (1995) Empirically defined regions of influence for clustering analyses. *Pattern Recognition*, 28(11), 1793 - 1806.

Overall, J. E. and Magee, K. N. (1992) Replication as a rule for determining the number of clusters in hierarchical cluster analysis. *Applied Psychological Measurement*, 16(2), 119 - 28.

Pollard, D. (1981) Strong consistency of k-means clustering. *The Annals of Statistics*, 9(1), 135 - 40.

Prim, R. C. (1957) Shortest connection networks and some generalizations. *Bell Sys. Tech. J.*, November, 1389 - 1401.

Qiu, G., Varley, M. R. and Terrell, T. J. (1994) Improved clustering using deterministic annealing with a gradient descent technique. *Pattern Recognition Letters*, 15, 607 - 10.

Ralambondrainy, H. (1995) A conceptual version of the k-means algorithm. *Pattern Recognition Letters*, 16, 1147 - 1157.

Rand, W. M. (1971) Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66(336), 846 - 50.

Rice, C. E. and Lorr, M. (1969) *An Empirical Comparison of Typological Analysis Methods*. (Contract N00014-67-A-0214), Office of Naval Research, Washington, D.C.

Ripley, B. D. (1992) Statistical Aspects of Neural Networks (pre-print). To appear in *Proceedings of Séminaire Européen de Statistique*, Sandbjerg, Denmark, Chapman and Hall, 1993.

Rubin, J. (1967) Optimal classification into groups: an approach for solving the taxonomy problems. *Journal of Theoretical Biology*, 15, 103.

Rumelhart, D. E. and Zipser, D. (1986) Feature discovery by competitive learning. In *Parallel Distributed Processing*, Vol. 1, Chap. 5, Cambridge Press, MIT.

Salzberg, S. L. (1999) On comparing classifiers: pitfalls to avoid and a recommended approach. To appear in: *Data Mining and Knowledge Discovery*, 1, 1-11.

Sarle, W. S. (1989) *SAS User's Guide: Statistics*, SAS Institute Inc., Cary, NC.

Scheibler D, and Schnieder, W. (1985) Monte Carlo tests of the accuracy of cluster analysis algorithms: a comparison of hierarchical and nonhierarchical methods. *Multivariate Behavioral Research*, 20, 283-304.

Selim, S. Z. and Alsultan, K. (1991) A simulated annealing algorithm for the clustering problem. *Pattern Recognition*, 24(10), 1003 - 8.

Shattuck, T., Germani, M. S. and Busek, P. R. (1991) Multivariate statistics for large data sets: Applications to individual aerosol particles. *Analytical Chemistry*, 63(22), 2646-2656.

Silverman, B. W. (1986) *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, London.

Sneath, P. H. A. (1957) The application of computers to taxonomy. *Journal of General Microbiology*, 17, 201 - 26.

Sokal, R. R. and Michener, C. D. (1958) A statistical method for evaluating systematic relationships, *University of Kansas Science Bulletin*, 38, 1409 - 38.

Srikanth, R., George, R., Warsi, N., Prabhu, D., Petry, F. E. and Buckles, B. P. (1995) A variable-length genetic algorithm for clustering and classification. *Pattern Recognition Letters*, 16, 789 - 800.

Sutanto, E. L., Mason, J. D. and Warwick, K. (1997) Mean-tracking clustering algorithm for radial basis function centre selection. *International Journal of Control*, 67(6), 961 - 977.

Tou, J. T. and Gonzalez, R. C. (1974) *Pattern Recognition Principals*, Addison-Wesley, Reading, MA.

Tyree E. W. and Long. J. A. (1998) A Monte Carlo evaluation of the moving method, k-means and self-organising neural networks. *Pattern Analysis and Applications*, 1(2).

Tyree E. W. and Long. J. A. (1997) Modelling clusters of arbitrary shape with Agglomerative - Partitional Clustering. In *First International Workshop on Statistical Techniques in Pattern Recognition '97*, Prague, Czech Republic.

Tyree E. W. and Long. J. A. (1996) Modelling clusters of arbitrary shape by agglomerative K-means clustering. *Equifax New Technology Forum: Decision Solutions*, Third Year, Fourth Workshop, North Leigh, UK.

Wallace, C. S. and Boulton, D. M. (1968) An information measure for classification. The *Computer Journal*, 11, 185 - 94.

Wang, Y., Yan, H and Sriskandarajah, C. (1996) The weighted sum of split and diameter clustering. *Journal of Classification*, 13, 231 - 248.

Ward, J. H. (1963) Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, 58, 236 - 44.

Wharton, S. W. (1984) An analysis of the effects of sample size on classification performance of a histogram based cluster procedure. *Pattern Recognition*, 17(2), 239-44.

Wishart, D. (1978) *Clustan User Manual*, 3rd Ed., Report No. 47, Program Library Unit, Edinburgh University.

Wishart, D. (1969a) Mode analysis. In *Numerical Taxonomy* (A. J. Cole, ed.), Academic Press, New York.

Wishart, D. (1969b) FORTRAN II programs for 8 methods of cluster analysis (CLUSTAN I). *Computer Contribution*, 38, State Geological Survey, Univ. of Kansas, Lawrence.

Wolfe, J. H. (1970) Pattern clustering by multivariate mixture analysis. Multivariate *Behavioral Research*, 5(3), 329-50.

Wong, Y-F (1993) Clustering by data melting. *Neural Computation*, 5, 89 - 104.

Wong, M. A (1982) A hybrid clustering method for identifying high-density clusters. *Journal of the American Statistical Society*, 22(380), 841 - 47.

Wong, M. A. and Lane, T. (1983) A kth nearest neighbour clustering procedure. *Journal of the Royal Statistical Society, Series B*, 45(3), 362 - 8.

Zahn, C. T. (1971) Graph theoretical methods for detecting and describing Gestalt clusters. *IEEE Transactions on Computers*, C-20(1), 68 - 86.

Zhang, Q. and Boyle, R. D. (1991) A new clustering algorithm with multiple runs of iterative procedures. *Pattern Recognition*, 24, (9), 835 - 848.