Subspace Identification with Cylindricity-Based Clustering for Hierarchical Fuzzy System Construction

Chong, A.[1] Gedeon, T.D.[1] Koczy, L.T.[1,2] Wong, K.W.[1]

[1]School of Information Technology
Murdoch University, Murdoch,
Western Australia 6150
Email:cchong@murdoch.edu.au

[2] Department of Telecom & Telematics
Budapest University of Technology and Economics
Email: koczy@ttt.bme.hu

Abstract: *Hierarchical fuzzy systems are proposed to deal with the rule explosion problem of traditional fuzzy systems. The inference operations of the fuzzy systems are well established. The next step is to tackle the problem of finding subspaces for automated hierarchical fuzzy system construction. In this paper, we propose a clustering technique designed specifically for this purpose. The technique exploits the concept of cylindricity and density to find subspace clusters for fuzzy systems construction. It is both theoretically and experimentally confirmed that the algorithm has reasonable accuracy and scalability.*

1. INTRODUCTION

Fuzzy systems suffer from rule explosion. To model a system with k variables and maximum T fuzzy terms in each dimension, the number of necessary rules is T^k, which will be very large if k is not very small. Because of this, fuzzy systems are limited to handle only very few variables.

Hierarchical fuzzy systems are designed to tackle this problem [1]. The idea is as follows. Often, the multi-dimensional input space $X = X_1 \times X_2 \times \ldots \times X_k$ can be decomposed into some subspaces, e.g. $Z_0 = X_1 \times X_2 \times \ldots \times X_{k0}$ ($k_0 < k$), so that in Z_0 a partition $\Pi = \{D_1, D_2, \ldots, D_n\}$ can be determined. In each D_i, a sub-rule base R_i can be constructed with local validity. The hierarchical rule base structure becomes:

R_0: *if* z_0 *is* D_1 *then use* R_1
if z_0 *is* D_2 *then use* R_2
⋮
if z_0 *is* D_n *then use* R_n

R_1: *if* z_1 *is* A_{11} *then y is* B_{11}
if z_1 *is* A_{12} *then y is* B_{12}
⋮
if z_1 *is* A_{1m1} *then t is* B_{1m1}

R_n: *if* z_n *is* A_{n1} *then y is* B_{n1}
if z_n *is* A_{n2} *then y is* B_{n1}
⋮
if z_n *is* A_{nmn} *then y is* B_{nmn}

The inference mechanism of the hierarchical fuzzy system has been established [1]. Hence, emphasis should now be placed on the construction of such hierarchical rule bases. The difficulty is mainly in finding the subspace Z_0 and the partition Π.

In this paper, we propose a clustering technique that is designed to find subspace clusters for hierarchical fuzzy system construction. Given a set of data, a clustering technique partitions the data into several groups such that the degree of association is strong within one group and weak for data in different groups. In the field of hierarchical fuzzy rule extraction, some special requirements on the clustering technique are introduced.

This paper is organized as follows. Section 2.0 discusses the clustering requirements. Section 3.0 presents the general problems of current clustering techniques. Subspace clusters are explained in section 4.0. This is followed by the discussion of CLIQUE [2], one of the earliest clustering techniques designed to find subspace clusters (section 5.0). A modified CLIQUE algorithm is presented in section 6.0. Section 7.0 discusses the weaknesses of the modified CLIQUE algorithm. This is followed by discussion of cylindricity in section 8.0. Our main proposed algorithm is presented in section 9.0. Section 10.0 is devoted to experiments that verify the effectiveness and efficiency of our proposed algorithm. This is followed by the conclustion (section 11).

2. CLUSTERING REQUIREMENTS

In this section, the requirements for clustering techniques designed for hierarchical fuzzy system construction are presented.

1. *Capable of handling high dimensional data.* The ultimate goal of hierarchical fuzzy systems is to break the limitation of fuzzy systems in the maximum number of variables that is manageable. If the goal is achieved, fuzzy systems may be used to model data with large numbers of dimensions. Hence, the clustering technique designed for the construction of hierarchical fuzzy systems must be able to handle high dimensional data.

2. *Interpretability of the clusters produced.* One of the advantages of fuzzy systems that distinguish it from neural networks is its ability to explain its inference results. Once a conclusion is reached, the user can observe the rules fired to gain insights into how and why the conclusion was reached. The interpretability of a fuzzy system relates directly to the fuzzy rules used. The clustering technique used to generate fuzzy rules should be designed to produce clusters that form easy-to-interpret fuzzy rules. This issue is addressed in more detail in section 3.0.

3. *No prior knowledge about data required.* Often, clustering techniques require certain input parameters from users. These techniques are usable in situations where the users possess prior information about the data being studied. One of the important goals of generic fuzzy systems modeling is to help users model a problem domain without requiring any prior knowledge about the domain. In this case, the clustering technique should not require prior knowledge about the data being studied from the user.

3. PROBLEMS WITH CURRENT CLUSTERING TECHNIQUES

In this section, the general problems in current clustering techniques in relation to fuzzy system construction are examined. A review of the literature suggests that no single clustering technique is designed to address all three requirements presented in section 2.0.

Let us first review the problem of high dimensionality. There are two valid reasons why distance function based clustering techniques can fail to cluster data with large numbers of dimensions. As the number of dimensions increases, the average density of points anywhere in the data space decreases. In this case, many dimensions or combination of dimensions can have noise or values that are uniformly distributed. This can cause distance functions that use all dimensions to fail. Beside this, it is often not meaningful to cluster data by examining the full data space, as most of the current clustering techniques do, since clusters may be embedded in certain subspaces (see section 4.0 for details).

The problem of high dimensionality also leads to the problem of computational feasibility. Some clustering algorithms are designed to identify clusters in high dimensional data [2, 3]. Unfortunately, their algorithms are too computationally complex. Although pruning strategies are introduced to reduce the complexity, the overall complexity of the techniques grows exponentially as the number of dimensions in the data increases. Hence, the scalability of such techniques is low.

Often, the limitation of current clustering techniques in dealing with high dimensional data is tackled in a few ways. A straightforward solution is to let the user specify the subspaces for cluster analysis [2]. This method is not only error-prone but also fails to address our third requirement presented in section 2.0. Another way to address high dimensionality is to apply a dimension reduction method to the data. Some examples of popular dimension reduction methods include principal component analysis (PCA) and Karhunen-Loeve transformation [4]. The basic idea of these techniques is to transform the original data space into a lower dimensional space by forming dimensions that are linear combinations of the individual original dimensions. While these methods are successful in dimension reduction, there are several disadvantages. Firstly, the dimension reduction comes with a price – loss of information. The greater the dimension reduction, the more information is lost and the less accurate the clusters become. Secondly, dimension reduction does not address the problem of finding clusters that exist in subspaces rather than the full space. Lastly and most important of all, dimension reduction methods introduce the problem of low interpretability in the clusters identified. By forming new dimensions, the user can no longer interpret the resulting clusters in relation to the original data space in a straightforward manner.

Some clustering techniques require prior knowledge about the data being studied. In the case of objective optimization algorithms, such as fuzzy c-means [5], the number of clusters in the data is required as an input parameter. For the purpose of generic fuzzy modeling, such an input parameter is unreasonable since the goal is often to create a fuzzy model based on the set of data where limited or no prior knowledge about the data is available.

4. SUBSPACE CLUSTERING

Let

$$X = \mathop{\mathrm{X}}_{i \in I} x_i \qquad \textbf{Eqn 4.1}$$

be the k dimensional data space, where I = { 1, 2, ..., k } is the set of dimension indexes. Then

$$S = \mathop{\mathrm{X}}_{i \in I_0} x_i \qquad \textbf{Eqn 4.2}$$

is a subspace of the full space, where $I_0 \subset I$. A subspace cluster is defined as a cluster that is embedded in a certain subspace. Figure 4.1 shows two one-dimensional subspace clusters embedded in

dimensions X1 and X2 respectively. Cluster C1 can be identified by observing its projection on X1. The data points in cluster C1 are spread uniformly across X2.

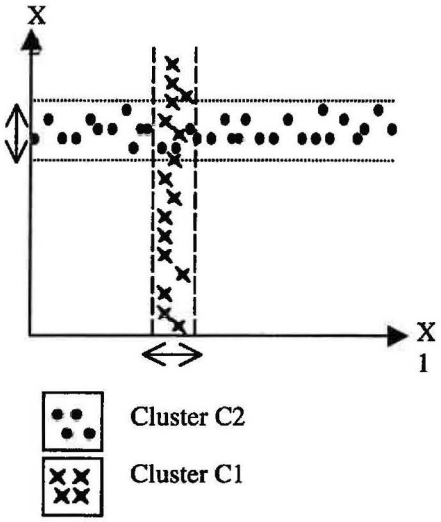

Figure 4.1: Subspace clusters C1 and C2 embedded in dimensions X1 and X2 respectively.

The existence of subspace clusters in data introduces new problems for distance function based clustering algorithms. Consider a set of data with dimensions $1...k$. If there exists a subspace cluster embedded in dimensions $1...k_0$ where k_0 is significantly smaller than k, then the data points in the cluster are distributed uniformly in dimensions $k_0...k$. In this case, it becomes difficult for distance functions that use all dimensions of the data to reflect the associations among data points within the cluster.

According to our review of the clustering literature, CLIQUE is the only clustering technique designed specifically to find subspace clusters. ENCLUST extends the idea of CLIQUE to exploit the concept of entropy. In this study, we modified CLIQUE to reduce the algorithm's computational complexity and exploit the concept of cylindricity (see section 8.0) to reduce the necessary input parameters to the clustering technique.

5. CLIQUE

In this section, the algorithm of CLIQUE [2] is discussed. The basic idea of CLIQUE is as follows. The multi-dimensional data space is first partitioned into non-overlapping hyperboxes. This is done by partitioning every dimension into ε number of equal-length intervals where ε is an input parameter. Each hyperbox is the intersection of one interval from each dimension. A data point is said to be contained in a hyperbox if its projections on all dimensions are within the intervals that comprise the hyperbox. A hyperbox is dense if the number of points in it exceeds a threshold t, which is another user set parameter. Similarly, a unit is defined to be the intersection of interval(s) from one or more dimensions.

Once all the dense units are found, clusters can be formed by connecting neighboring units. The core of the clustering technique lies in the algorithm to identify dense units. The algorithm is based on the Apriori algorithm [6] used extensively in data mining. The algorithm proceeds in multiple passes. In the first pass, all the one-dimensional units are examined and the dense units become candidates to the next pass. In general, having determined (k-1) dimensional dense units, the candidate k-dimensional units are determined using the candidate generation procedure given below.

C_k = set of candidates at pass k
$u.a_i$ = i^{th} dimension of unit u
$u.[l_i,h_i)$ = interval in the i^{th} dimension
D_{k-1} = set of all (k-1) dimensional dense units

insert into C_k
select $u_1.[l_1,h_1)$, $u_1.[l_2,h_2)$, ..., $u_1.[l_{k-1},h_{k-1})$, $u_2.[l_{k-1},h_{k-1})$
from D_{k-1} u_1 D_{k-1} u_2
where $u_1.a_1 = u_2.a_1$, $u_1.l_1 = u_2.l_1$, $u_1.h_1 = u_2.h_1$,
$u_1.a_2 = u_2.a_2$, $u_1.l_2 = u_2.l_2$, $u_1.h_2 = u_2.h_2$, ...,
$u_1.a_{k-2} = u_2.a_{k-2}$, $u_1.l_{k-2} = u_2.l_{k-2}$, $u_1.h_{k-2} = u_2.h_{k-2}$,
$u_1.a_{k-1} = u_2.a_{k-1}$, $u_1.l_{k-1} = u_2.l_{k-1}$, $u_1.h_{k-1} = u_2.h_{k-1}$

The relation < represents lexicographic ordering on dimensions. Upon candidate generation, dense units that have a projection in (k-1)-dimensions that are not included in C_{k-1} are discarded. The resulting candidates then go through the MDL-based pruning stage.

Given the subspaces s_1, s_2, ..., s_n, the MDL-based technique first groups together the dense units that lie in the same subspace. Then for each subspace, the coverage is computed as:

$$coverage \quad (s_j) = \sum_{u_i \in s_j} count \quad (u_i) \qquad \textbf{Eqn 5.1}$$

Where count(u_i) is the number of points that is contained in u_i. Subspaces with small coverage are pruned. The algorithm terminates when no more candidates are left for a particular pass.

Using a bottom-up approach and discarding non-dense units in the early passes, the algorithm is able to prune a significant volume of the error space. The MDL-based pruning method further discards candidates that are less likely to be clusters, increasing the speed of the algorithm. We remark that the MDL-based pruning method can be error prone. Figure 5.1 shows situation where MDL-based pruning can be ineffective. In the figure, the bold edged units are more likely to be retained than those real cluster units due to their high coverage.

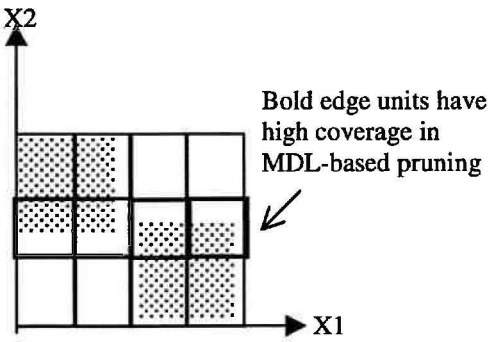

Figure 5.1: Converage of units.

Even with the pruning strategies introduced in [2], the algorithm still suffers from high computational complexity. This is explained as follows. If a dense unit exists in k-dimensions, then all of its projections in a subset of k-dimensions is also dense. The total number of combinations to be explored by the algorithm to identify the dense unit is calculated as

$$\sum_{i=1}^{k}\binom{k}{i} \quad where \quad \binom{k}{i}=\frac{k!}{k(k-i)!} \qquad \textbf{Eqn 5.2}$$

The overall complexity of the algorithm is thus, $O(c^k)$ for some constant c. Therefore, improvement on the algorithm to reduce the computational complexity is necessary. In the next section, we present our modified algorithm with reduced complexity.

6. FAST CLIQUE

One of the goals of the proposed clustering technique is to produce clusters for the construction of hierarchical fuzzy system. Since fuzzy rules operate on the projections of the multi-dimensional clusters, convex clusters are desired for the fuzzy system generation. Hence, one of the differences between our algorithm and the original CLIQUE is that our algorithm is designed to approximate convex clusters.

The basic idea of CLIQUE is retained in our modified algorithm. The algorithm starts by partitioning every dimension into ε (user parameter) number of equal-length intervals. A unit is considered dense if the number of data points contained in the unit exceeds the threshold t (user parameter). The algorithm consists of the following four steps.

1. Find one of the dense units, u, that exceeds the threshold t.
2. Approximate convex cluster C, by expanding the dense unit u in each of the dimensions that the dense unit is embedded in.
3. Remove all data points that are contained in the cluster C as just approximated.
4. Repeat steps 1 – 3 until no dense unit can be found.

The pseudo-C-code for the important procedures involved in the algorithm is presented.

```
PROCEDURE find_dense_unit
        Let U_i be the set of one-dimensional units in
        dimension i
        Let denseunit = [ ]
        for i = 1 to k
          for each unit u ∈ U_i
            utemp = denseunit x u
            if utemp is dense
              denseunit = utemp
              break
            end if
          end for
        end for
```

For the convenience of discussion, we define [ ] as the zero-dimensional (empty) subspace where $[\ ] \times X_i = X_i$. This procedure scans through each of the dimensions to find one of the dense units in the data.

```
PROCEDURE approximate_convex_cluster(u)
Let D be the set of dimension indexes of u
for each i in D
  Let clusterset = { }
  expand_along(u,i)

Let ul = left most element of clusterset along
dimension i
Let ur = right most element of clusterset along
dimension i

  u.l = ul.l
  u.h = ur.h
end for
```

Given a dense unit, the procedure approximate_convex_cluster expands the unit along all the dimensions that the unit is embedded in. The procedure results in a hyper-rectangular cluster.

```
PROCEDURE expand_along(u,i)
global clusterset
add u to clusterset

Let u^l = left neighboring unit of u along dimension i
if u^l is dense
  expand_along(u^l,i)
end if

Let u^r = right neighboring unit of u along dimension i
if u^r is dense
  expand_along(u^r,i)
end if
```

The procedure expand_along is used by approximate_convex_cluster to expand a dense unit along a certain dimension. Using the modified algorithm, the computational complexity is greatly reduced. To find a dense unit that exists in k-dimensions, the procedure find_dense_unit performs a single pass through each dimension of the data, giving the complexity O(k). To approximate a convex cluster using the k-dimensional dense unit, the procedure approximate_convex_cluster examines each of the k dimensions $O(k)$ by calling the procedure expand_along. The procedure expand_along examines both the right and left neighboring units. In the worse case, all ε number of units are examined $O(\varepsilon)$. The entire algorithm terminates when all clusters are found. Thus the overall complexity is:

$$O(\ c \times (k + k\varepsilon)\ ) = O(ck\varepsilon) \qquad \textbf{Eqn 6.1}$$

Since the complexity of the algorithm is linear, it is computationally feasible to cluster data with very large numbers of dimensions. This is a marked improvement over the original algorithm.

7. USER PARAMETER THRESHOLD

In this section, we examine in detail the important user input parameter to our algorithm – threshold t. Recall from section 6.0 that we make use of the threshold to find dense units. The accuracy of our algorithm relies heavily on the threshold selected. A high threshold causes the algorithm to undesirably miss out some dense units while a low threshold results in misidentifying noise as clusters. For the convenience of discussion, we define cluster units as units that contains data points from clusters and noise units as units that contains noise data points.

An ideal threshold is one that can be used to distinguish dense units from noise units. Hence, proper estimation of the threshold requires prior information such as the percentage of noise or the number of data points in the least dense cluster about the data being studied. Despite the fact that the prior information is often not available to the user, there are other problems with threshold estimation. The rest of this section discusses the difficulties of this process.

In the presence of a subspace cluster, it is not possible to accurately identify the percentage of noise data points in a set of data. Consider a set of k-dimensional data containing a subspace cluster embedded in the dimensions 1,2, ..., k_0 where $k_0 < k$. Then the data points contained in the cluster can be considered as cluster data points in dimensions 1,2, ..., k_0 but become noise data points in dimensions k_0+1, k_0+2, ..., k-1, k. Therefore, threshold estimation based on the percentage of noise in a set of data can be error prone.

It is more accurate to estimate the threshold based on the number of points contained in the least dense cluster in the data. Let C be the least dense cluster in the data and assume that points are uniformly distributed in clusters, the threshold t can be estimated as follows.

t = number of points in C / number of units that can fit into cluster C **Eqn 7.1**

Expanding the equation, we have:

$$t = \frac{n/p}{\prod_{i \in D} f(s_i/b_i)} \qquad \textbf{Eqn 7.2}$$

Where $f()$ can be the ceiling or floor function (more details later),

n = number of data points in C
p = total number of data
D = set of dimension indexes
s_i = length of cluster C in the i^{th} dimension
b_i = length of units in the i^{th} dimension

Figure 7.1 shows the effects of choosing ceiling or floor as the function $f()$ in the equation.

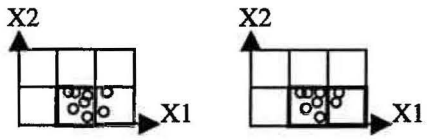

Figure 7.1 (a) Coverage of cluster when floor is chosen as function $f()$ in equation 7.2 (b) Coverage of cluster when ceiling is chosen as function $f()$ in equation 7.2

Although eqn 7.2 provides a reasonable estimate of threshold t when prior information about the least dense cluster is known, there are situations where no single threshold exists to separate cluster units and noise units. Consider figure 7.2.

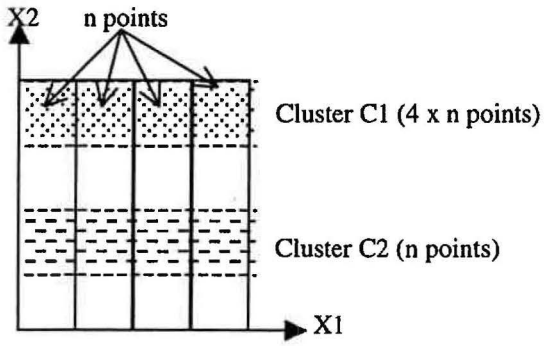

Figure 7.2: Subspace clusters with different density

In the figure, both clusters are subspace clusters embedded in dimension X2. The user input ε is chosen as 4. Cluster C2 is the least dense cluster. Using the technique discussed so far, our threshold t will be selected as n. Each of the two dimensional unit in cluster C2 has n/4 number of points. Hence, none of the 2D units is dense. Projecting the points in C2 onto dimension X2, a one-dimensional dense unit is obtained. Therefore, the subspace cluster C2 embedded in dimension X2 is successfully identified.

Now consider cluster C1 in the figure. Since it is four times more dense than cluster C2, the 2D units in C1 have n points each. So all the 2D units are considered dense. This results in the algorithm misidentifying cluster C1 as a two dimensional cluster (the 2D units will be merged to form a single unit by the algorithm eventually). In this case, no individual threshold t can be used by the algorithm to identify the two one-dimensional subspace clusters.

The example above shows the disadvantage of using a density threshold for cluster analysis. It is clear that criteria other than density are needed for successful cluster analysis. Consider again figure 7.2, where although all the two-dimensional units in cluster C1 exceed the threshold density, the fact that the cluster points are spread uniformly across dimension X1 suggests that it is not embedded in X1. At this stage, we wish to bring the concept of 'cylindricity' into our discussion. In our example, the cluster C1 is said to be 'completely cylindrical' along dimension X1. In the next section, the concept of cylindricity is discussed in depth.

8. THE CONCEPT OF CYLINDRICITY

If a cluster c contains points that are spread uniformly along a certain dimension X_i, then the cluster is indistinguishable from its projection along the dimension. This is because observing the cluster projection along the dimension does not provide us with any information. In this case, we call cluster c cylindrical along dimension X_i.

An example of a subspace cluster was given in section 2.0. Here we discuss the relation between cylindricity and subspace clusters. Consider a set of k-dimensional data containing a subspace cluster embedded in the dimensions 1,2, ..., k_0 where $k_0 < k$, then the cluster is cylindrical along the dimensions k_0+1, k_0+2, ..., k-1, k.

In more general terms, cylindricity can be defined as the extent to which data points are uniformly distributed. In the rest of this section, we discuss a method to measure cylindricity. A number of statistical techniques can be used for this purpose. A careful examination of the techniques reveals that chi square test [7] can be easily integrated into our clustering algorithm.

Consider partitioning a dimension into n intervals of fixed width. The cylindricity of a set of data points along the dimension can be computed by examining their projections to the dimension. If the set of points is completely cylindrical along the dimension, then the number of points projected into each of the n intervals should be the same as shown in figure 8.1a.

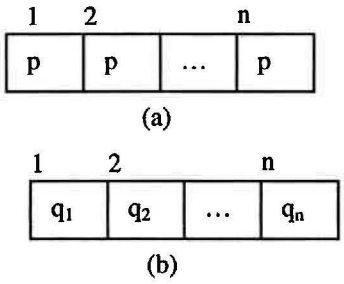

Figure 8.1 (a) expected number of projected points (b) realistic number of projected points

Complete cylindricity is rare in practice. Suppose the observed number of points is as in Figure 8.1b. Cylindricity can be measured as the extent to which the observed numbers (figure 8.1b) differs from the set of numbers with complete cylindricity (figure 8.1a). The difference can be calculated as:

$$err = \sum_{i=1}^{n} \frac{q_i - p}{q_i} \qquad \textbf{Eqn 8.1}$$

It is statistically proven that eqn 8.1 resembles a chi square distribution with the degree of freedom $n - 1$. The cylindricity can then be computed as:

$$P(\chi^2(n\text{-}1) > err) \qquad \textbf{Eqn 8.2}$$

Eqn 8.2 models cylindricity as a probability. There are two advantages to this scheme:

1. The scheme allows for normalized values [0 – 1].
2. Since there are well-established statistical significance thresholds for a variety of disciplines, it is easy to determine whether the cylindricity is significant.

9. CYLINDRICITY-BASED SUBSPACE CLUSTERING

In this section, we proposed a cylindricity-based subspace-clustering algorithm. The basic idea of our technique is the following. Cylindricity is used to determine whether there is any cluster embedded in a dimension. Using figure 7.2 as an example, the

algorithm starts by computing the cylindricity of dimension X1. Because there is no cluster embedded in the dimension, the resulting cylindricity will be significantly high. Therefore, the algorithm will ignore the dimension and move on to examine the next dimension, X2. Here there are two clusters and the cylindricity will be significantly low. The algorithm will then pick the unit with highest density which, in this case, is the unit contained in cluster C1.

The complete algorithm is discussed in the rest of this section. The algorithm starts by partitioning every dimension into ε (input parameter) number of equal-length intervals. The four-step algorithm is presented following by the pseudo-C-code of the important procedures used.

1. Find one of the dense units u using procedure find_dense_unit
2. Approximate convex cluster C, by expanding the dense unit u in each of the dimensions in which the dense nit u is embedded.
3. Remove all data points that are contained in cluster C and *disable all units found to be part of the cluster.*
4. Repeat steps 1 – 3 until no dense unit can be found.

PROCEDURE find_dense_unit

```
Let U_i be the set of one-dimensional units in dimension i
Let denseunit = [ ]
for i = 1 to k
   Let cyl = cylindricity of U_i
   if cyl is significantly low
    Select u ∈ U_i such that denseunit x u is most dense
     denseunit = denseunit x u
   end if
end for
```

PROCEDURE expand_along(*u*,i)

```
global clusterset
add u to clusterset

Let u^r = right neighboring unit of u along dimension i
if is_cluster(u^r,i)
   expand_along(u^r,i)
end if

Let u^l = left neighboring unit of u along dimension i
if is_cluster(u^l,i)
   expand_along(u^l,i)
end if
```

PROCEDURE is_cluster(*u*,i)

```
Let cyl1 = cylindricity of dimension i
Let cyl2 = cylindricity of dimension i disregarding u

if cyl2 is significantly higher than cyl2
   return true
else
   return false
end if
```

PROCEDURE approximate_convex_cluster(*u*)

```
Let D be the set of dimension indexes of u
for each i in D
   Let clusterset = { }
   expand_along(u,i)

Let ul = left most element of clusterset along dimension i
Let ur = right most element of clusterset along dimension i

   u.l = ul.l
   u.h = ur.h
end for
```

The algorithm proposed in this section has two main extensions to the algorithm in section 6.0. Firstly the concept of disabling units is introduced in the cylindricity-based algorithm. In general, any unit that has been disabled will no longer take part in the algorithm. Recall from the algorithm that once a cluster is found, all the data points contained in the cluster will be removed. This procedure results in all units contained by the cluster to be empty. These empty units will have negative impact to our calculation of cylindricity in the upcoming steps. Disabling the units can prevent them from affecting the accuracy of our algorithm.

Secondly, dense units are no longer determined based on a density threshold. The concept of cylindricity has been used for two purposes. In the procedure find_dense_unit, cylindricity is used to determine whether there is any cluster embedded in a dimension. In the procedure is_cluster, cylindricity is used to determine whether a unit is a noise or cluster unit. The idea is that if the unit is a cluster unit, then removing it from the dimension should increase the cylindricity of the dimension significantly.

10. EXPERIMENTAL RESULTS

In this section, the performance of the proposed cylindricity-based clustering technique is evaluated. Both synthetic generated data and real world data have been used to verify the effectiveness and efficiency of the algorithm.

10.1 Synthetic Generated Data

In this section, the overall goal of the experiments is to evaluate the efficiency and accuracy of the algorithm. In terms of efficiency, the experiments aim to determine the scalability of the algorithms in:

1. The dimensionality of the data space
2. The dimensionality of clusters
3. Number of data

The experiments have been structured similarly to [2] so that comparison between CLIQUE and our algorithms can be made.

The synthetic data generator from [8] is used to produce the data for the experiments. Figure 10.1 shows a sample input to the data generator to generate two dimensional data. In the figure, cluster 2 and 3 are subspace clusters embedded in dimension X2 and X1 respectively. A total of 3300 data points are generated in which 10% of the data is noise.

Cluster	X1	X2	Number of Points
1	[0.2, 0.3)	[0.2, 0.3)	10000
2	[0.0, 1.0)	[0.5, 0.6)	10000
3	[0.8, 0.9)	[0.0, 1.0)	10000
Noise	[0.0, 1.0)	[0.0, 1.0)	3000

Figure 10.1 Sample input to data generator

The clusters generated are hyper-rectangles in shape and data points are uniformly distributed within the clusters. The rest of this section discusses the results of the experiments. Figure 10.2b, 10.3b, and 10.4b are extracted from [2] to allow for comparisons.

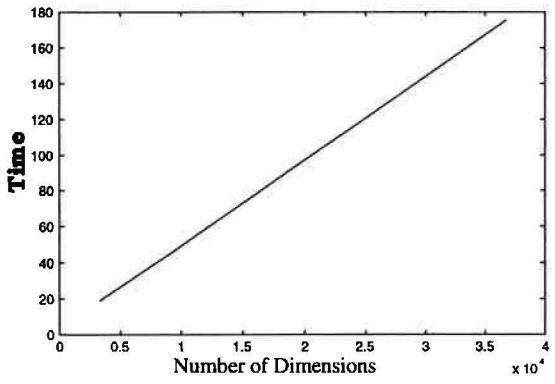

Figure 10.2a Scalability with the number of data

Figure 10.2a shows the scalability of our algorithm as the number of data increases. The data has five dimensions. There are three clusters embedded in the full space. The number of data increases from 3300 to 368000. From the figure, our algorithm scales linearly with the increase of the number of data. It is shown in figure 10.2b that CLIQUE performs equally well.

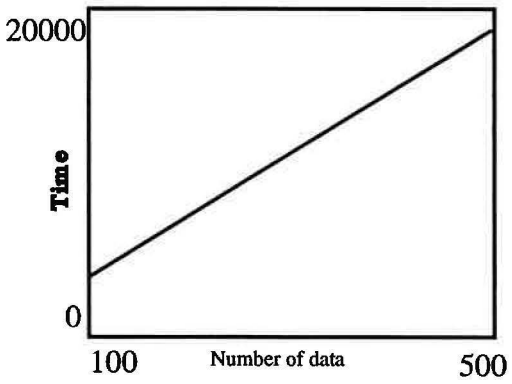

Figure 10.2b CLIQUE scalability with the number of data

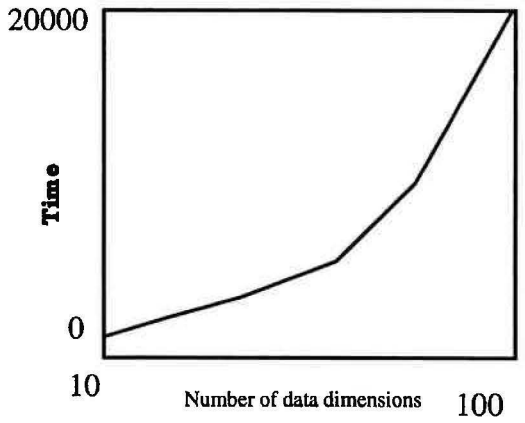

Figure 10.3b CLIQUE scalability with the data space dimensionality

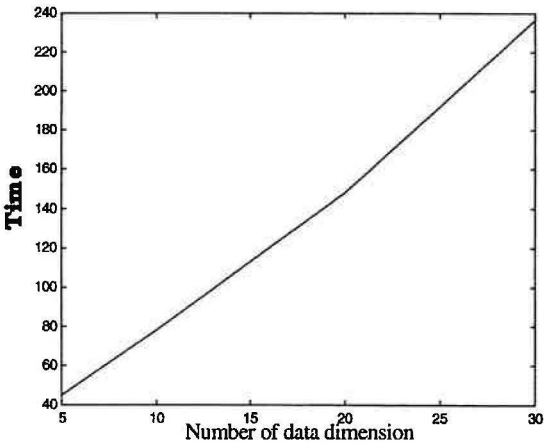

Figure 10.3a Scalability with the data space dimensionality

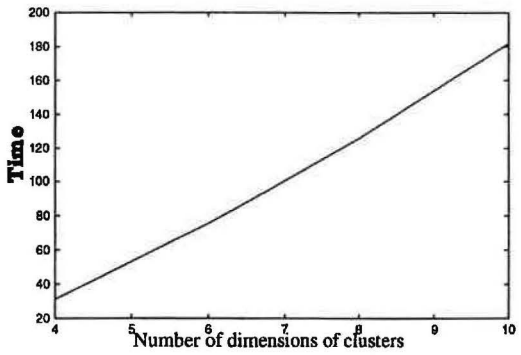

Figure 10.4a Scalability with the cluster dimensionality

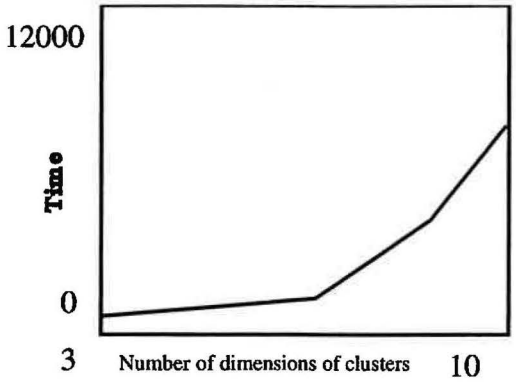

Figure 10.4b CLIQUE scalability with the cluster dimensionality

Figure 10.3a shows the scalability of our algorithm as the number of dimensions of the data increases. There are three five dimensional subspace clusters. There are 9150 number of data points. The number of dimensions of the data space increases from 5 to 30 dimensions. Again, our algorithm scales linearly with the increase of the data space dimensionality. Figure 10.3b shows a quadratic or worse behavior for CLIQUE.

Figure 10.4a shows the scalability of our algorithm as the highest dimensionality of the clusters increases. The number of dimensions increases from 4 to 10 dimensions. Our algorithm scales linearly with the increase of cluster dimensionality. The performance of CLIQUE is quadratic or worse as shown in figure 10.4b.

In all the above experiments, our algorithm is able to recover the original clusters in the data. Apart from that, it is easily observed that the algorithm scales linearly with the increase of the dimensionality of the data space, the dimensionality of clusters as well as the number of data. Overall, the experiments show that our algorithm outperforms CLIQUE significantly in every case.

10.2 Real World Data

In this section, we apply our algorithm to a set of real-life data. The US 1990 census data (person record), obtained from http://www.ipums.umn.edu is used. The 16-dimensional data consists of 10,000 points. Table 10.1 lists some of the meaningful subspaces found by our algorithm.

FAMSIZE NCHILD NCHILD5 AGE CHBORN YRIMMIG SPEAKENG
FAMSIZE NCHILD NCHILD5 CHBORN YRIMMIG SPEAKENG
FAMSIZE NCHILD NCHILD5 ELDCH
FAMSIZE NCHILD NCHILD5 AGE
FAMSIZE NCHILD NCHILD5 CHBORN
FAMSIZE NCHILD

Table 10.1a Subspaces found by the algorithm

FAMSIZE NCHILD NCHILD5
FAMSIZE NCHILD5 YRIMMIG SPEAKENG
FAMSIZE YRIMMIG SPEAKENG INCBUS
YRIMMIG SPEAKENG INCTOT INCBUS
NCHILD5 YRIMMIG SPEAKENG INCTOT INCBUS
CHBORN YRIMMIG SPEAKENG UHRSWORK INCBUS
FAMSIZE NCHILD NCHILD5 YRIMMIG SPEAKENG INCBUS
NCHILD NCHILD5 YRIMMIG SPEAKENG UHRSWORK INCBUS
NCHILD NCHILD5 YRIMMIG SPEAKENG INCTOT INCBUS

Table 10.1b Subspaces found by CLIQUE

Legend

FAMSIZE	Number of own family members in household
NCHILD	Number of own children in household
NCHILD5	Number of own children under age 5
ELDCH	Age of youngest own child in household
AGE	Age
CHBORN	Number of children ever born
YRIMMIG	Year of immigration
SPEAKENG	Speaks English
UHRSWORK	Usual hours worked per week
INCTOT	Total personal income
INCBUS	Non-farm business income

Table 10.1c Legend

It can be seen that FAMSIZE, NCHILD and NCHILD5 are predominant in the results. This is understandable since the number of family members,

children and children under the age of 5 should be highly correlated. YRIMMIG and SPEAKENG are found in the first and second rows of the table. Their correlation can be explained by the fact that one of the important criteria for migrating to the US is the ability to speak English. Detailed examination of the result reveals that our algorithm is able to discover meaningful subspaces.

Figure 10.2 shows the subspaces found by CLIQUE. Similar to our algorithm, it is observed that FAMSIZE, NCHILD, and NCHILD5 has shown high correlations in the results. YRIMMIG and SPEAKENG are also highly correlated. The main difference in the two sets of results is that there are a number of subspaces (e.g. UHRSWORK, INCTOT, INCBUS) which are found in CLIQUE but not in our algorithm. This can be attributed to the fact that CLIQUE identifies subspace clusters based on density regardless of the cylindricity. There may be subspaces whose density exceeds the threshold but are actually cylindric. These subspaces will be ignored by our algorithm. Apart from that, the number of subspaces identified by CLIQUE is highly sensitive to the threshold selected.

11. CONCLUSION

In this paper, we have proposed a subspace-clustering algorithm. The algorithm is designed to find convex subspace clusters that can be used for the construction of hierarchical fuzzy systems. Our algorithm is an improvement over CLIQUE, one of the first clustering techniques designed to find subspace clusters. It was both theoretically and experimentally confirmed that the complexity of our algorithm is significantly reduced. Since the computational complexity of our algorithm is low, it can used to deal with high dimensional data. The clustering technique brings us one step nearer to the efficient automatic construction of hierarchical fuzzy systems.

The concept of cylindricity has been defined and used in the proposed clustering technique. The use of cylindricity has not only improved the effectiveness of the algorithm, but also reduced the number of necessary user parameters to the technique. The next steps are to verify the reliability of the algorithm using a wider range of real life data, and explore the use of the algorithm to construct hierarchical fuzzy systems.

12. REFERENCES

[1] Koczy, L.T. *Approximative inference in hierarchical structured rule bases.* in *Fift IFSA World Congress.* 1993. Seoul: International Fuzzy Systems Association.

[2] Agrawal, R., Gehrke, J., Gunopulon, D., and Raghawan, P. *Automatic subspace clustering of high dimensional data for data mining applications.* in *Proceedings of the ACM SIGMOD conference on Management of Data.* 1998. Canada.

[3] Cheng, C.H., Fu, A.W., and Zhang, Y. *Entropy-based Subspace Clustering for Mining Numerical Data.* in *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-99).* 1999. San Diego.

[4] Jackson, J.E., *A User's Guide To Principal Components.* 1991, US: John Wiley & Sons.

[5] Bezdek, J.C., *Pattern Reconition with Fuzzy Objective Function Algorithms.* 1981, New York: Plenum Press.

[6] Agrawal, R. and Srikant, R. *Fast algorithms for mining association rules.* in *Proceedings of the 20th VLDB Conference.* 1994.

[7] Weiss, A.N., *Elementary Statistics.* 3 ed. 1996: Addison-Wesley.

[8] Zait, M. and Messatfa, H., *A Comparative Study of Clustering Methods.* Future Generation Computer Systems, 1997. 13: p. 149-159.