CHAPTER 6

An Efficient Deterministic psFCM Clustering Algorithm

This chapter presents an efficient deterministic version of the psFCM clustering algorithm [111]. The chapter also presents experimental results, obtained by applying the proposed algorithm on a number of benchmark data sets, to illustrate its efficiency over the pshFCM [112], psFCM and FCM clustering algorithms.

6.1 Introduction

The Fuzzy c-means algorithm (presented in section 4.5), like its hard version the K-means algorithm, belongs to the family of local search algorithms, which searches for the solution optimum using a greedy (hill-climbing) approach. As such, the local search algorithms, like FCM, often fail to search for the global optimum. Also, the execution of FCM algorithm frequently involves the creation of a large number of membership matrices and candidate cluster matrices. The FCM algorithm is thus a computationally intensive method. The execution time of the FCM algorithm can be reduced, if a good set of initial cluster centroids is chosen, because the algorithm will take less number of iterations to find the actual cluster centroids. It is however difficult to select a good set of initial cluster centroids randomly [112]. Thus, in this chapter, an efficient deterministic fuzzy clustering algorithm is presented that obtains better initial centroids through partitioning of the data set using a variation of the k-d tree space-partitioning data structure [113]. Like the HBDKM algorithm described in chapter 5, the proposed fuzzy clustering algorithm, presented in this chapter, also
obtains the initial centroids in a deterministic way. Thus the overall clustering obtained by the proposed algorithm is also deterministic. The algorithm is presented in detail in section 6.4. In section 6.2, a brief outline of some related work on seed initialization of FCM is presented. In section 6.3, the k-d tree data structure is explained along with its variants. In section 6.5 experimental results for enunciating the efficiency of the proposed approach is presented. Finally, in section 6.6, a summary of the chapter is provided.

6.2 Related Work

The FCM clustering algorithm is the most widely used and researched approach in both theoretical and practical applications of fuzzy logic to data clustering. Many FCM based clustering algorithms have been proposed in the literature. Cheng et al. [114] proposed a multistage random sampling FCM algorithm. It is based on the assumption that a small subset of a data set of feature vectors can be used to approximate the cluster centroids of the complete data set. With this assumption, FCM is used to compute the cluster centroids of a small randomly selected subset of the original data set. After obtaining the cluster centroids of this small subset, the subset is merged with an additional small, randomly selected subset of the remaining unprocessed feature vectors to form a large subset to be processed by FCM. The previously calculated cluster centroids are used for the initialization of the fuzzy partition matrix of the newly formed set. The above procedure is repeated until the size of the feature vector matrix used in calculations is large enough to approximate the actual cluster centroids of the full data set. The resulting cluster centroids are then used for the initialization of the fuzzy partition matrix for FCM when it is applied to the original data set. Hung and Yang [111] proposed an efficient FCM clustering algorithm called psFCM \textit{(partition simplification FCM)} algorithm. It is divided into two phases. In Phase I, the dataset is first divided into some hyper-blocks using the k-d tree data structure. All patterns in a hyper-block are then replaced by the centroids of the patterns. In this way the original dataset is drastically cut down to a simplified dataset comprising of small number of hyper-blocks' centroids. The FCM algorithm is
then applied on this simplified dataset to find the actual initial cluster centroids for the complete dataset. In Phase II, the FCM algorithm is applied on the complete data set using the optimal seeds obtained from Phase I. Begum et al. [112] also presented a $k$-d tree based divisive technique for seed initialization for the FCM clustering algorithm. The approach has three phases. Phase I partitions the data set into several hyper-blocks using the $k$-d tree partitioning method and then computes the centroids of all the obtained hyper-blocks using an approach similar to that in Phase I of [111]. In Phase II a set of $c$ centroids are selected at random from the set of all computed centroids obtained in Phase I and are evaluated using the popular, fuzzified $PBM$ validity index, $PBMF$. This process is continued for a specific number of times for different randomly selected sets and the set of seeds corresponding to the highest value of $PBMF$ index is chosen as the optimal set of seeds for clustering the complete data set. Using these seeds, the FCM algorithm is then applied on the complete data set, to obtain the final clustering in Phase III.

### 6.3 The $k$-d tree Data Structure

The $k$-d tree data structure is a popular space-partitioning data structure for organizing points in a $k$-dimensional space. It is a data structure that subdivides the hyperspace into hyper-rectangular cells (called here as hyper-blocks) through the recursive application of some splitting rule. The choice of splitting rule affects the shape of hyper-blocks and the structure of the resulting tree. The $k$-d tree is a generalization of the simple 1-dimensional binary search tree in which every node is associated with one of the $k$ dimensions. In a $k$-d tree, every non-leaf node of the tree can be thought of as implicitly generating a splitting hyperplane that divides the hyperspace of $k$-dimensional patterns into two parts, known as subspaces. Points to the left of this hyperplane represent the left sub-tree of that node and points to the right of the hyperplane represent the right sub-tree. The root node of the tree represents the entire $k$-dimensional data set [113] [115] [116] [117]. The idea of $k$-d trees was first introduced by Bentley [113] in 1975. As per the original definition, if a $k$-dimensional data set is represented as a $k$-d tree, then each pattern in the data set is
stored as a node in the tree. Each node contains two pointers, which are either null or point to another node in the k-d tree. Each pointer can be considered as specifying a sub-tree. Each node at a particular depth in the tree is associated with a discriminator dimension (not necessarily stored as a field in the node), which is an integer in the interval \([1, k]\) corresponding to the \(k\) dimensions. The original notion of a k-d tree was however modified by Friedman et al. in [116] by introducing a k-d tree which stores data only in the leaf nodes, often as small collections of patterns from the data set called buckets (here hyper-blocks). The two notions were later termed by Bentley as a homogeneous and a nonhomogeneous k-d tree respectively, in [117]. Thus a homogenous k-d tree is a binary tree in which every node stores a k-dimensional data pattern from the data set whereas a nonhomogenous k-d tree is a binary tree which stores data points only in the leaf nodes, often as collections of several k-dimensional points from the data set, in hyper-blocks. A non-homogeneous k-d tree is also sometimes referred to as a k-d trie or a pseudo k-d tree [118].

Given a list of \(n\), k-dimensional points, the following algorithm will construct a k-d tree containing those points.

**Table 6.1: Procedure to construct a k-d tree**

<table>
<thead>
<tr>
<th>Procedure split (pointList, depth)</th>
</tr>
</thead>
<tbody>
<tr>
<td>// as per some splitting rule select a discriminator dimension and return a</td>
</tr>
<tr>
<td>// discriminator value for that dimension (usually median or midpoint)</td>
</tr>
<tr>
<td>end split</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Procedure kdtree (pointList, depth)</th>
</tr>
</thead>
<tbody>
<tr>
<td>if pointList is empty</td>
</tr>
<tr>
<td>return null;</td>
</tr>
<tr>
<td>else</td>
</tr>
<tr>
<td>discriminator = split(pointList, depth)</td>
</tr>
<tr>
<td>// Create a tree node, node and construct sub-trees</td>
</tr>
<tr>
<td>node.discriminator = discriminator;</td>
</tr>
<tr>
<td>node.leftChild = kdtree(points in pointList less than discriminator, depth+1);</td>
</tr>
<tr>
<td>node.rightChild = kdtree(points in pointList greater than or equal to discriminator, depth+1);</td>
</tr>
<tr>
<td>return node;</td>
</tr>
<tr>
<td>end if</td>
</tr>
</tbody>
</table>

The procedure split has been defined in several ways since its first definition by Bentley in [113]. In the next subsection some popular splitting rules are presented.
including the original splitting rule presented in [113] by Bentley.

### 6.3.1 Splitting Rules

In this subsection, some well-known splitting rules for the construction of the \( k \)-d tree space partitioning data structure are presented. The splitting rule used for the construction of \( k \)-d tree structure by the proposed algorithm of this chapter, is also presented. A splitting rule obtains the discriminator dimension, i.e., the dimension which should be split into two subspaces, and the discriminator value \( \text{discriminator} \) for each node in a \( k \)-d tree.

**The original split rule:** The original splitting rule for \( k \)-d tree construction proposed by Bentley [113] chooses the discriminator dimension for each node on the basis of its depth in the tree. The root node is defined to be at depth 0. The discriminator dimension is calculated as \( \text{depth} \mod k + 1 \); this formula results in cycling through the \( k \) dimensions. The discriminator value for a chosen discriminator dimension is given by the coordinate median of the points in that dimension.

**Standard split:** Instead of cycling through the dimensions, this rule selects the discriminator dimension for each node to be the one for which the current \( \text{pointList} \) have the maximum spread value (difference between the maximum and minimum values along the dimension). The discriminator value is chosen to be the coordinate median of the points in that dimension. Friedman, Bentley and Finkel introduced this splitting rule in their definition of the optimized \( k \)-d tree in [116]. This rule is now referred to as the standard splitting rule.

**Midpoint split:** As per this rule, the splitting hyperplane passes through the center of the hyper-block and bisects the longest side of the hyper-block. If there are many sides of equal length, any one may be chosen first, say the one with the lowest coordinate index. The rule differs from the standard split in the way that it uses the midpoint, rather than the median, as the \( \text{discriminator} \). Thus the splitting hyperplane need not always pass through a data point [115].

**Sliding-midpoint split:** In this rule, first a midpoint split is attempted, by considering a hyperplane passing through the center of the hyper-block and bisecting the hyper-block's longest side. If the data points lie on both sides of the splitting
6.4 The proposed DpsFCM Clustering Algorithm

The proposed DpsFCM clustering algorithm consists of two phases which are analogous to that of the HBDKM algorithm discussed in chapter 5. Phase I is the

Hyperplane: The proposed DpsFCM clustering algorithm combines the advantages of deterministic psFCM and sliding hyperblock (SHB) clustering. In this section, the proposed DpsFCM clustering algorithm is presented. The algorithm consists of two phases which are analogous to the HBDKM algorithm discussed in chapter 5. Phase I is the

Hybrid split: This rule is a hybridization of the original split rule, the standard split rule and the midpoint split rule. According to this rule, initially a sorted one dimensional array disc_dim_nos of the k dimension numbers, 1 to k, is computed by sorting the numbers in decreasing order according to the spread values (difference between the maximum and minimum values along a dimension) of the k dimensions for the complete data set. The rule then selects the discriminator dimension numbers for the nodes at various depths by cycling through the k dimension numbers from the sorted list disc_dim_nos. At a particular depth, depth, the index of the discriminator dimension number in disc_dim_nos is given by depth mod k + 1. After the selection of the discriminator dimension the rule continues with the splitting using the midpoint split rule. The root node is defined to be at depth 0.

From the description it can be easily seen that the hybrid split rule is designed to be computationally fast by:

- restricting the spread value calculation to only one instance, unlike that in the standard, midpoint and sliding-midpoint rules which require computation after each split for choosing the next dimension to be split.
- using the simple rule for choice of the discriminator dimension based on depth from disc_dim_nos, similar to that of the original split rule.
- avoiding the costly calculation of mean and using the midpoint instead.

The proposed algorithm presented in this chapter thus uses this hybrid split rule.

6.4 The proposed DpsFCM Clustering Algorithm

In this section the proposed DpsFCM (deterministic psFCM) clustering algorithm is presented. The proposed DpsFCM algorithm consists of two phases which are analogous to that of the HBDKM algorithm discussed in chapter 5. Phase I is the

Hyperplane: Then, the splitting hyperplane remains here. However, if all the data points lie to one side of the splitting hyperplane, then splitting hyperplane 'slides' towards the data points until it encounters the first point. A child leaf hyper-block containing this single point is created, and the algorithm reiterates on the remaining points. In all splits the rule uses the midpoint, rather than the median, as the discriminator [115].

Hybrid split: This rule is a hybridization of the original split rule, the standard split rule and the midpoint split rule. According to this rule, initially a sorted one dimensional array disc_dim_nos of the k dimension numbers, 1 to k, is computed by sorting the numbers in decreasing order according to the spread values (difference between the maximum and minimum values along a dimension) of the k dimensions for the complete data set. The rule then selects the discriminator dimension numbers for the nodes at various depths by cycling through the k dimension numbers from the sorted list disc_dim_nos. At a particular depth, depth, the index of the discriminator dimension number in disc_dim_nos is given by depth mod k + 1. After the selection of the discriminator dimension the rule continues with the splitting using the midpoint split rule. The root node is defined to be at depth 0.

From the description it can be easily seen that the hybrid split rule is designed to be computationally fast by:

- restricting the spread value calculation to only one instance, unlike that in the standard, midpoint and sliding-midpoint rules which require computation after each split for choosing the next dimension to be split.
- using the simple rule for choice of the discriminator dimension based on depth from disc_dim_nos, similar to that of the original split rule.
- avoiding the costly calculation of mean and using the midpoint instead.

The proposed algorithm presented in this chapter thus uses this hybrid split rule.

6.4 The proposed DpsFCM Clustering Algorithm

In this section the proposed DpsFCM (deterministic psFCM) clustering algorithm is presented. The proposed DpsFCM algorithm consists of two phases which are analogous to that of the HBDKM algorithm discussed in chapter 5. Phase I is the
6.4 The proposed DpsFCM Clustering Algorithm

initialization phase that involves the fast and deterministic computation of seeds and checking of their fitness. Phase II is the clustering phase in which the FCM algorithm is applied only 'once' on the whole data set using the optimal set of centroids obtained in Phase I. The complete algorithm is now presented next:

**Phase I**

Step 1): Obtain the number of clusters $c$.

Step 2): Set maximum depth of partitioning, $maxd = \lceil \log_2(c) \rceil + 1$.

Step 3): Compute the spread values (difference between the maximum and minimum values along a dimension) of the $k$ dimensions for the complete data set. Create the sorted one dimensional array $disc\_dim\_nos$ of the $k$ dimension numbers, 1 to $k$, by sorting the numbers in decreasing order according to the spread values.

Step 4): Partition the dataset $X$ by simulating the creation of a nonhomogeneous $k$-d tree up to the maximum depth $maxd$, starting at depth 0, via hybrid split, using the array $disc\_dim\_nos$, so that an $M \times (k+1)$ matrix $T$ is obtained. The matrix $T$ contains $M (\leq 2^{maxd})$ centroids $x_m$'s and the $M$ counts of patterns $n_m$'s for all the $M$ non-empty hyper-blocks generated temporarily in the process of partitioning of $X$ into a $k$-d tree with $M$ leaf nodes. Each leaf node represents a hyper-block of $k$-dimensional patterns from the original data set. (Procedure given in Table 6.2).

If $M \geq c$ then

Step 5): Sort the $M$ centroids in $T$, in decreasing order according to the values $n_m$'s of the densities of their corresponding hyper-blocks and choose top $c$ centroids and store them in optimal centroid set $cen\_opt1$.

Step 6): Obtain another set of optimal initial centroids from $T$ using a procedure described in Table 6.3 and store it in optimal centroid set $cen\_opt2$. [In each iteration of the procedure, it calculates the pair-wise distances of each pair of centroids present in $T$, sorts the distances in increasing order, computes the mean of two closest centroids in the list of centroids and removes the two centroids. It then adds the mean of the
centroids as a new centroid in the list of centroids. After $M - c$ iterations the optimal centroid set $cen_{opt2}$ is obtained.

**Step 7)**: Evaluate the value of the fitness measure, the fuzzy PBM validity index, $PBMF$ (described in section 4.6.2) for the optimal centroid set $cen_{opt1}$ and store the value obtained in $pbmfval1$.

**Step 8)**: Evaluate the value of the fuzzy PBM validity index, $PBMF$ for the optimal centroid set $cen_{opt2}$ and store the value obtained in $pbmfval2$.

**Step 9)**: Compare the two values $pbmfval1$ and $pbmfval2$ and select the optimal centroid set corresponding to the larger of the two values as the final optimal centroid set $cen_{opt}$.

Else set $cen_{opt}$ as an empty set and terminate

**Phase II**

**Step 10)**: Apply FCM algorithm on the original data set only once, using the optimal set of centroids $cen_{opt}$ obtained in Step 9 and output the result of clustering.

As can be seen from the algorithm, Step 1 of Phase I obtains the value of $c$. Step 2 calculates the maximum depth of partitioning $maxd$. Step 3 obtains the sorted array $disc_{dim\_nos}$, by sorting the $k$ dimensions numbers in decreasing order according to the spread values. In Step 4, the hyper-volume of the data set $X$ is partitioned using the procedure described in Table 6.2 as a result of which the original dataset $X$ is drastically cut down to a very simplified form $T$ comprising of only a small number ($\leq 2^{maxd}$) of hyper-blocks' centroids. In Step 5, the centroids in $T$ are sorted according to the densities of the $M$ hyper-blocks and the $c$ centroids corresponding to the top $c$ dense hyper-blocks are stored in the optimal centroid set $cen_{opt1}$. In Step 6, another optimal centroid set $cen_{opt2}$ is obtained using the procedure described in Table 6.3. In steps 7 and 8 the $PBMF$ index values for the two centroid sets $cen_{opt1}$ and $cen_{opt2}$ are calculated and in Step 9, the centroid set corresponding to the higher value of the $PBMF$ validity index is chosen as the optimal centroid set $cen_{opt}$. Step 5 through Step 9 are executed only if the number of leaf nodes $M$ (the number of rows in $T$) is greater than or equal to $c$ else the algorithm sets $cen_{opt}$ to an empty matrix and terminates.
6.4 The proposed DpsFCM Clustering Algorithm

Table 6.2: Procedure to partition a data set \( X \) by simulating the creation of a nonhomogeneous \( k \)-d tree up to a maximum depth \( \text{maxd} \) starting at \( \text{depth} = 0 \)

Procedure `kdtreepart` \((X, \text{depth}, \text{maxd}, \text{disc_dim_nos})\)

// Calculate the number of rows \( n \) and number of columns \( k \), of \( X \)
if \( n = 0 \)
   \( T = \) an empty matrix
   return \( T \);
else if \( n > 1 \) and \( \text{depth} < \text{maxd} \)
   // Select the discriminator dimension based on \( \text{depth} \) by cycling through all
   // the values in \( \text{disc_dim_nos} \). Select the discriminator value as the midpoint
   \( \text{disc_dim} = \text{disc_dim_nos}(\text{depth} \mod k + 1) \)
   // Find the maximum and minimum values along the dimension \( \text{disc_dim} \)
   \( \text{max_disc_dim} = \) the maximum value of dimension \( X(:, \text{disc_dim}) \)
   \( \text{min_disc_dim} = \) the minimum value of dimension \( X(:, \text{disc_dim}) \)
   \( \text{discriminator} = (\text{min_disc_dim} + \text{max_disc_dim})/2 \)
   \( X_1 = \) points in \( X \) less than discriminator
   \( X_2 = \) points in \( X \) greater than or equal to discriminator
   // Calculate the number of rows \( s_1 \) of \( X_1 \) and the number of rows \( s_2 \) of \( X_2 \)
   if \( s_1 > 0 \)
      \( T_1 = \) `kdtreepart`(\( X_1, \text{depth}+1, \text{maxd}, \text{disc_dim_nos} \));
   end if
   if \( s_2 > 0 \)
      \( T_2 = \) `kdtreepart`(\( X_2, \text{depth}+1, \text{maxd}, \text{disc_dim_nos} \));
   end if
   // Calculate the number of rows \( s_{11} \) of \( T_1 \) and the number of rows \( s_{22} \) of \( T_2 \)
   if \( s_{11} > 0 \) and \( s_{22} > 0 \)
      \( T = \) a matrix obtained by concatenating \( T_1 \) and \( T_2 \)
   elseif \( s_{11} > 0 \) and \( s_{22} = 0 \)
      \( T = T_1 \);
   elseif \( s_{11} = 0 \) and \( s_{22} > 0 \)
      \( T = T_2 \);
   else
      if \( n > 1 \)
         \( T(1, 1:k) = \) mean of \( X \)
      else
         \( T(1, 1:k) = X \)
      end if
      \( T(1, k+1) = n \)
      return \( T \)
   end if
end kdtree
6.4 The proposed DpsFCM Clustering Algorithm

In Step 10 of Phase II the FCM clustering algorithm is applied on the complete data set by initializing the pseudopartition matrix of the FCM algorithm using the optimal initial centroids $cen_{opt}$ obtained in Step 9 of Phase I.

Table 6.3: Procedure to obtain the optimal centroid set $cen_{opt2}$ from the $M \times (k+1)$ matrix $T$

<table>
<thead>
<tr>
<th>Procedure optcenset2 $(T)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt; Initialize the matrix of optimal centroid set $cen_{opt2}$</td>
</tr>
<tr>
<td>$cen_{opt2} = T(:,1:k)$ // an $M \times k$ matrix</td>
</tr>
<tr>
<td>&gt; Initialize the matrix of optimal centroid set $cen_{opt2}$</td>
</tr>
<tr>
<td>$dif = M - c$</td>
</tr>
<tr>
<td>&gt; Calculate the number of rows $ncen$ in the matrix $cen_{opt2}$</td>
</tr>
<tr>
<td>&gt; Create a matrix $dist_cen$ with $\frac{ncen(ncen-1)}{2}$ rows and 3 columns</td>
</tr>
<tr>
<td>&gt; Sort the rows of $dist_cen$ in ascending order according to the distance &gt; values in the 3rd column, of the centroid pairs</td>
</tr>
<tr>
<td>$cen1 = cen_{opt2}(dist_cen(1,1),:\cdot)$ // Top two closest centroids are</td>
</tr>
<tr>
<td>$cen2 = cen_{opt2}(dist_cen(1,2),:\cdot)$ // stored in $cen1$ and $cen2$</td>
</tr>
<tr>
<td>&gt; Delete the two centroids $cen1$ and $cen2$ from the matrix $cen_{opt2}$</td>
</tr>
<tr>
<td>$cen = mean\ of\ the\ centroids,\ cen1\ and\ cen2$</td>
</tr>
<tr>
<td>$cen_{opt2} = a\ matrix\ obtained\ by\ concatenating\ cen_{opt2}\ and\ cen$</td>
</tr>
<tr>
<td>end for</td>
</tr>
<tr>
<td>return $cen_{opt2}$</td>
</tr>
<tr>
<td>end optcenset2</td>
</tr>
</tbody>
</table>

It can now be seen that the proposed DpsFCM algorithm involves no randomness, as the choice of good seeds is done deterministically rather than resorting to random techniques. The algorithm generates only two candidate sets of seeds and chooses the better one by evaluating the efficient $PBMF$ validity index, thus the quality of clustering is also ensured. It needs to apply FCM algorithm on the full data
set only once rather than for an uncertain number of times. The average execution time taken by the whole process is thus comparatively quite less. The final outcome of clustering by the DpsFCM algorithm is also highly optimal.

## 6.5 Experimental Results and Discussion

To demonstrate the better performance of the proposed DpsFCM algorithm over the pshFCM, psFCM and FCM algorithms, a comparative study was carried out by applying the four algorithms on all the twelve benchmark data sets described in subsection 4.7.1. For all the algorithms the validation of clustering was done using the efficient \( PBMF \) validity index. For the sake of uniform comparison the same partitioning procedure, presented in Table 6.2, was also used for the pshFCM and psFCM algorithms. The experimental results obtained using the algorithms are presented in Tables 6.4 through 6.8 and in Figures 6.1 through 6.9. In Table 6.4, the values obtained by the four algorithms for the \( PBMF \) validity index, corresponding to their optimal set of initial centroids, are compared and in Table 6.5, the values obtained by the four algorithms for the \( PBMF \) validity index, corresponding to their final optimal clustering results, are compared. In Table 6.6 and Table 6.7, the iteration counts and average execution times (respectively) of the four algorithms are compared. Due to the non-deterministic nature of the pshFCM, psFCM and FCM algorithms, the algorithms were run for 50 trials on each of the data sets. For FCM, the iteration counts and average execution times corresponding to the first best trial with optimal clustering (indicated by the lowest value of misclassification error percentage or highest value of the \( PBMF \) index), encountered during the 50 trials, are presented and for the pshFCM and psFCM algorithms the iteration counts and average execution times corresponding to the first best trial with optimal seeds (indicated by the highest value of the \( PBMF \) index), encountered during the 50 trials, are presented. The total execution times of all trials for the pshFCM, psFCM and FCM algorithms for each of the data sets are presented in Table 6.8. For the DpsFCM clustering algorithm also, the average execution times (as presented in Table 6.7) were obtained by executing the algorithm for 50 trials on all twelve data sets.
Table 6.4: Comparison of values obtained for \(PBMF\) validity index by DpsFCM, pshFCM and psFCM clustering algorithms for their corresponding optimal set of initial centroids, for the actual number of clusters in all data sets

<table>
<thead>
<tr>
<th>Data set</th>
<th>Actual no. of clusters</th>
<th>(PBMF) index value obtained for initial centroids by</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iris</td>
<td>3</td>
<td>22.5997, 24.8601, 13.8843</td>
</tr>
<tr>
<td>Cancer</td>
<td>2</td>
<td>98.8271, 176.7426, 92.0861</td>
</tr>
<tr>
<td>Wine</td>
<td>3</td>
<td>293386.9594, 331575.3169, 278771.5960</td>
</tr>
<tr>
<td>SODAR1</td>
<td>3</td>
<td>29198.8489, 35213.9468, 28361.3310</td>
</tr>
<tr>
<td>SODAR2</td>
<td>3</td>
<td>56576.6911, 56576.6911, 30472.2335</td>
</tr>
<tr>
<td>Data_3_2</td>
<td>3</td>
<td>17.4287, 17.9150, 16.3502</td>
</tr>
<tr>
<td>Data_5_2</td>
<td>5</td>
<td>22.9278, 23.7360, 22.8135</td>
</tr>
<tr>
<td>Data_6_2</td>
<td>6</td>
<td>619.6013, 554.0484, 619.6217</td>
</tr>
<tr>
<td>Data_9_2</td>
<td>9</td>
<td>6.853, 7.2971, 6.7458</td>
</tr>
<tr>
<td>Data_10_2</td>
<td>10</td>
<td>308.7863, 305.6905, 310.3326</td>
</tr>
<tr>
<td>Data_4_3</td>
<td>4</td>
<td>954.9403, 954.9403, 800.6757</td>
</tr>
<tr>
<td>Ruspini</td>
<td>4</td>
<td>21278.9258, 24463.3192, 21880.5954</td>
</tr>
</tbody>
</table>

Table 6.5: Comparison of values obtained for \(PBMF\) validity index by DpsFCM, pshFCM and psFCM clustering algorithms for final clustering results, for the actual number of clusters in all data sets

<table>
<thead>
<tr>
<th>Data set</th>
<th>Actual no. of clusters</th>
<th>(PBMF) index value obtained for final clustering by</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iris</td>
<td>3</td>
<td>28.2205, 28.2205, 28.2204</td>
</tr>
<tr>
<td>Cancer</td>
<td>2</td>
<td>167.964, 167.964, 167.964</td>
</tr>
<tr>
<td>Wine</td>
<td>3</td>
<td>555537, 555537, 555537</td>
</tr>
<tr>
<td>SODAR1</td>
<td>3</td>
<td>34745.1, 34745.1, 34745.1</td>
</tr>
<tr>
<td>SODAR2</td>
<td>3</td>
<td>58914.4, 58914.4, 58914.3</td>
</tr>
<tr>
<td>Data_3_2</td>
<td>3</td>
<td>16.6513, 16.6513, 16.6513</td>
</tr>
<tr>
<td>Data_5_2</td>
<td>5</td>
<td>22.9332, 22.9332, 22.9332</td>
</tr>
<tr>
<td>Data_6_2</td>
<td>6</td>
<td>638.347, 638.347, 638.347</td>
</tr>
<tr>
<td>Data_9_2</td>
<td>9</td>
<td>6.78327, 6.78327, 6.78327</td>
</tr>
<tr>
<td>Data_10_2</td>
<td>10</td>
<td>345.917, 345.917, 345.918</td>
</tr>
<tr>
<td>Data_4_3</td>
<td>4</td>
<td>960.79, 960.79, 960.79</td>
</tr>
<tr>
<td>Ruspini</td>
<td>4</td>
<td>25196.7, 25196.7, 25196.7</td>
</tr>
</tbody>
</table>
The misclassification error percentages were obtained as same for all the four algorithms. For all the four algorithms and for the *PBMF* index, \( m \) was chosen as 1.5.

**Table 6.6: Comparison of iteration counts of one deterministic trial of DpsFCM with those for the first best trials of pshFCM, psFCM and FCM algorithms for obtaining the final clustering for the actual number of clusters in all the data sets**

<table>
<thead>
<tr>
<th>Data set</th>
<th>Actual__no.__of__clusters</th>
<th>No.__of__iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iris</td>
<td>3</td>
<td>DpsFCM 17</td>
</tr>
<tr>
<td>Cancer</td>
<td>2</td>
<td>DpsFCM 9</td>
</tr>
<tr>
<td>Wine</td>
<td>3</td>
<td>DpsFCM 25</td>
</tr>
<tr>
<td>SODAR1</td>
<td>3</td>
<td>DpsFCM 8</td>
</tr>
<tr>
<td>SODAR2</td>
<td>3</td>
<td>DpsFCM 5</td>
</tr>
<tr>
<td>Data_3_2</td>
<td>3</td>
<td>DpsFCM 5</td>
</tr>
<tr>
<td>Data_5_2</td>
<td>5</td>
<td>DpsFCM 22</td>
</tr>
<tr>
<td>Data_6_2</td>
<td>6</td>
<td>DpsFCM 4</td>
</tr>
<tr>
<td>Data_9_2</td>
<td>9</td>
<td>DpsFCM 23</td>
</tr>
<tr>
<td>Data_10_2</td>
<td>10</td>
<td>DpsFCM 14</td>
</tr>
<tr>
<td>Data_4_3</td>
<td>4</td>
<td>DpsFCM 4</td>
</tr>
<tr>
<td>Ruspini</td>
<td>4</td>
<td>DpsFCM 6</td>
</tr>
</tbody>
</table>

**Table 6.7: Comparison of average execution times of one trial of DpsFCM, pshFCM, psFCM and FCM algorithms for obtaining the final optimal clustering results, for the actual number of clusters in all the data sets**

<table>
<thead>
<tr>
<th>Data set</th>
<th>Actual__no.__of__clusters</th>
<th>Average execution time (in seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iris</td>
<td>3</td>
<td>DpsFCM 0.010108</td>
</tr>
<tr>
<td>Cancer</td>
<td>2</td>
<td>DpsFCM 0.017161</td>
</tr>
<tr>
<td>Wine</td>
<td>3</td>
<td>DpsFCM 0.017368</td>
</tr>
<tr>
<td>SODAR1</td>
<td>3</td>
<td>DpsFCM 0.0049287</td>
</tr>
<tr>
<td>SODAR2</td>
<td>3</td>
<td>DpsFCM 0.0041282</td>
</tr>
<tr>
<td>Data_3_2</td>
<td>3</td>
<td>DpsFCM 0.0040322</td>
</tr>
<tr>
<td>Data_5_2</td>
<td>5</td>
<td>DpsFCM 0.002786</td>
</tr>
<tr>
<td>Data_6_2</td>
<td>6</td>
<td>DpsFCM 0.014587</td>
</tr>
<tr>
<td>Data_9_2</td>
<td>9</td>
<td>DpsFCM 0.16861</td>
</tr>
<tr>
<td>Data_10_2</td>
<td>10</td>
<td>DpsFCM 0.084626</td>
</tr>
<tr>
<td>Data_4_3</td>
<td>4</td>
<td>DpsFCM 0.010908</td>
</tr>
<tr>
<td>Ruspini</td>
<td>4</td>
<td>DpsFCM 0.0045701</td>
</tr>
</tbody>
</table>
Table 6.8: Comparison of total execution times of 50 trials for pshFCM, psFCM and FCM algorithms, for the actual number of clusters in all the data sets

<table>
<thead>
<tr>
<th>Data set</th>
<th>Actual no. of clusters</th>
<th>Total execution time (in seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iris</td>
<td>3</td>
<td>pshFCM: 0.73454, psFCM: 0.56267, FCM: 0.57225</td>
</tr>
<tr>
<td>Cancer</td>
<td>2</td>
<td>pshFCM: 1.5056, psFCM: 0.7689, FCM: 0.77189</td>
</tr>
<tr>
<td>Wine</td>
<td>3</td>
<td>pshFCM: 1.0611, psFCM: 0.98498, FCM: 0.78153</td>
</tr>
<tr>
<td>SODAR1</td>
<td>3</td>
<td>pshFCM: 0.37563, psFCM: 0.26468, FCM: 0.21005</td>
</tr>
<tr>
<td>SODAR2</td>
<td>3</td>
<td>pshFCM: 0.33214, psFCM: 0.19427, FCM: 0.17542</td>
</tr>
<tr>
<td>Data_3_2</td>
<td>3</td>
<td>pshFCM: 0.33485, psFCM: 0.15655, FCM: 0.28596</td>
</tr>
<tr>
<td>Data_5_2</td>
<td>5</td>
<td>pshFCM: 1.8181, psFCM: 1.3231, FCM: 1.3952</td>
</tr>
<tr>
<td>Data_6_2</td>
<td>6</td>
<td>pshFCM: 3.4853, psFCM: 1.98, FCM: 1.6639</td>
</tr>
<tr>
<td>Data_9_2</td>
<td>9</td>
<td>pshFCM: 15.5528, psFCM: 10.2675, FCM: 11.8722</td>
</tr>
<tr>
<td>Data_10_2</td>
<td>10</td>
<td>pshFCM: 8.3531, psFCM: 5.9341, FCM: 6.1877</td>
</tr>
<tr>
<td>Data_4_3</td>
<td>4</td>
<td>pshFCM: 1.182, psFCM: 0.56814, FCM: 0.81325</td>
</tr>
<tr>
<td>Ruspini</td>
<td>4</td>
<td>pshFCM: 0.34937, psFCM: 0.1986, FCM: 0.25091</td>
</tr>
</tbody>
</table>

Figures 6.1 through 6.9 show the results obtained by the DpsFCM algorithm, in graphical manner. Each figure corresponding to a data set shows the optimal initial centroids and the best final clustering obtained by the algorithm for the data set. For all the data sets in 2-dimensional and 3-dimensional space, the $k$-d tree partitioning is also shown. Graphical results are not shown for data sets with $d > 3$.

Figure 6.1: (a) Optimal initial centroids obtained for the SODAR1 data set with $maxd = 3$, (b) Final clustering obtained for SODAR1 data set.
6.5 Experimental Results and Discussion

Figure 6.2: (a) Optimal initial centroids obtained for the SODAR2 data set with maxd = 3, (b) Final clustering obtained for SODAR2 data set

Figure 6.3: (a) Optimal initial centroids obtained for the Data_3_2 data set with maxd = 3, (b) Final clustering obtained for Data_3_2 data set

Figure 6.4: (a) Optimal initial centroids obtained for the Data_5_2 data set with maxd = 4, (b) Final clustering obtained for Data_5_2 data set
6.5 Experimental Results and Discussion

Figure 6.5: (a) Optimal initial centroids obtained for the Data_6_2 data set with \( maxd = 4 \), (b) Final clustering obtained for Data_6_2 data set

Figure 6.6: (a) Optimal initial centroids obtained for the Data_9_2 data set with \( maxd = 5 \), (b) Final clustering obtained for Data_9_2 data set

Figure 6.7: (a) Optimal initial centroids obtained for the Data_10_2 data set with \( maxd = 5 \), (b) Final clustering obtained for Data_10_2 data set
6.5 Experimental Results and Discussion

Figure 6.8: (a) Optimal initial centroids obtained for the Data_4_3 data set with $maxd = 3$, (b) Final clustering obtained for Data_4_3 data set

Figure 6.9: (a) Optimal initial centroids obtained for the Ruspini data set with $maxd = 3$, (b) Final clustering obtained for Ruspini data set

From the results presented in Table 6.4, it can be clearly inferred that the proposed DpsFCM algorithm is quite efficient in obtaining very good initial centroids for FCM clustering compared to the non-deterministic psFCM and pshFCM clustering algorithms. Figures 6.1 through 6.9 clearly show that the initial centroids that are obtained by the proposed DpsFCM algorithm are indeed very good and in almost all of the cases the seeds are obtained quite near to the final centroids of the actual clusters. It is important to note that the DpsFCM algorithm obtains the seeds in a deterministic way unlike the psFCM and pshFCM algorithms. Thus, the set of optimal seeds obtained by the DpsFCM algorithm is fixed for each data set. As such, for the DpsFCM algorithm, the values of the $PBMF$ validity index, for both $cen\_opt$ and for final clustering results for a given data set, also remain the same and the number of
iterations of the only FCM trial required, is also fixed. The DpsFCM algorithm is thus fully deterministic, and the final result of clustering is also very optimal (as shown in Table 6.5). From Table 6.6 it can be easily inferred that the numbers of iterations required by the DpsFCM algorithm are also very optimal and are comparable to the iteration counts of best trials of pshFCM, psFCM and FCM algorithms. For the non-deterministic algorithms, pshFCM, psFCM and FCM, the iteration counts presented in Table 6.6 can frequently vary due to their inherent random nature, but for DpsFCM, the iteration counts are fixed. From Table 6.7, it can be seen that the DpsFCM algorithm is comparable in terms of average execution time with the non-deterministic pshFCM, psFCM and FCM clustering algorithms. For some data sets like Iris, Data_9_2 and Data_4_3, the DpsFCM algorithm executes much faster because of the requirement of lesser number of iterations for convergence. For other data sets, one deterministic trail of the DpsFCM algorithm takes slightly more time than the other algorithms. However, from Table 6.8 it can be seen that the total execution times of all the random number (here 50) of trials required for obtaining the best clustering by each of non-deterministic algorithms pshFCM, psFCM and FCM, for each of the data sets, can be much more than the average execution time of one deterministic trial of the proposed DpsFCM clustering algorithm.

For obtaining the time complexity of the DpsFCM clustering algorithm, the following main time complexities need to be considered:

(i) The time complexity of exploring a data set to find the maximum and minimum values along each dimension is $O(n)$.

(ii) The time complexity of creating the array $\text{disc\_dim\_nos}$ is $O(k \log k)$.

(iii) The time complexity of partitioning of a data set by building a $k$-d tree is $O(kn \log n)$ [116].

(iv) The time complexity of computing the first set of seeds is $O(c \log c)$.

(v) The time complexity of computing the second set of seeds is $O(c^3)$.

(vi) In Phase II of the DpsFCM algorithm the FCM algorithm is executed once. One trial of FCM has the time complexity $O(nc k t)$.

Thus the total time complexity of the DpsFCM algorithm turns out to be:

$$O(n) + O(k \log k) + O(kn \log n) + O(c \log c) + O(c^3) + O(nc k t)$$
For data sets with $k, c \ll n$, the complexity turns out to be:

$$O(n) + O(kn \log n) + O(nckt)$$

The DpsFCM algorithm is thus slightly costly than the FCM algorithm, but is comparable in complexity with the other two algorithms, pshFCM and psFCM, which also use the $k$-d partitioning technique. The experimental results however, enunciate that the degree of certainty of good clustering is very high for the DpsFCM algorithm compared to all the other non-deterministic algorithms, pshFCM, psFCM and FCM, where there is no guarantee that in a specified number of trials, the algorithms will output the most optimal clustering for a given data set.

From all the results presented in Tables 6.4 through 6.8 and Figures 6.1 through 6.9 and from the time complexity analysis, it can thus be inferred that the DpsFCM deterministic clustering algorithm indeed performs better than the other non-deterministic clustering algorithms, with which it was compared. The performance of DpsFCM algorithm can be further improved by using a kernel version of the $PBMF$ validity index named $kPBMF$ index, which is described in detail in chapter 7.

### 6.6 Chapter Summary

In this chapter a fully deterministic version of the psFCM clustering algorithm is presented. The algorithm finds better initial centroids for clustering compared to the pshFCM, psFCM and FCM clustering algorithms. To illustrate the efficiency of the proposed algorithm it was compared with the pshFCM, psFCM and FCM algorithms through several experiments on a number of benchmark data sets and from the experimental results it can be concluded that the DpsFCM algorithm shows superior clustering performance over the other clustering algorithms in terms of the determination of initial centroids, number of trials and iterations and total execution time.