Fuzzy Clustering Method for Content-based Indexing

K.S. Leung, I. King and H.Y. Yue
Department of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin, New Territories, Hong Kong
Email: {ksleung, king, hyyue}@cse.cuhk.edu.hk

Abstract

Efficient and accurate information retrieval is one of the main issues in multimedia databases. In content-based multimedia retrieval databases, contents or features of the database objects are used for retrieval. To retrieve similar database objects, we often perform a nearest-neighbor search. A nearest-neighbor search is used to retrieve similar database objects with features nearest to the query under the feature vector space with a given distance function (similarity measurement). Typically, data exist in natural cluster. However, many of the currently indexing methods do not utilize this data cluster information in the construction of the indexing structure which leads to performance degradation. To improve the retrieval performance, we (1) use *Fuzzy Competitive Clustering (FCC)*, a noise resistance fuzzy clustering algorithm, to locate good approximate cluster prototypes efficiently, (2) use the result of FCC clustering to construct a good indexing structure (FCC-b-tree) for effective nearest-neighbor search and (3) Dervied two elimination rules for purning the indexing tree in searching process. Our experimental results show that: (1) FCC gets the better cluster prototypes then other traditional clustering algorithms in general. and (2) The FCC-b-tree always has a better performance than linear search.

1 Introduction

Efficient and accurate information retrieval is one of the main issues in multimedia databases. The content-based retrieval lets users to specify queries by features (or contents) such as *color*, *texture*, *sketch* and *shape* to retrieve database objects with features similar to the queries.

Many content-based retrieval multimedia database systems have been developed in the past few years. For example, Montage [1] is an image database for managing and retrieving visual information efficiently and effectively. It supports content-based retrieval by *color histogram*, *sketch*, *texture*, and *shape*. Query by Image Content (QBIC) [2] allows queries on databases based on color, texture, and shape of database objects. Photobook [3] makes use of semantics-preserving image compression to support search based on three image content descriptions: appearance, 2-D shape, and textural properties. VisualSEEk [4] is a content-based image and video retrieval system for World Wide Web. It uses color contents and the spatial layout of color regions of images for retrieval. Other multimedia database systems which support content-based query include Chabot [5], MMIS [6], VIMSYS [7], ART MUSEUM [8], KMeD [9], and CORE [10]. Although the above databases use different features for retrieval, most of them have shown that they are efficient and effective.

In a typical multimedia database, all database objects have to be pre-analyzed and then organized in a special way for retrieval. The main steps are:

1. **Feature Extraction -** The corresponding features from each of the database objects are first extracted. These features are usually stored in the form of real-valued multi-dimensional vectors.

2. **Index Structure Building -** The database may then organize the extracted features by using an indexing structure for retrieval.

3. **Content-Based Retrieval -** Content-based retrieval can be performed on the indexing structure efficiently and effectively.

1.1 Nearest-neighbour Searching

By using feature vectors, the content-based retrieval multimedia databases support similar searching. Applying a suitable distance function to the feature vectors as the similarity measurement, the database objects will be ranked according to a query. The top ranked objects are then retrieved as the result for similar retrieval. Nearest-neighbor search is a typical kind of similar searching. In the feature vector space or the real space, a query can be seen as a multi-dimensional point (or vector). Thus, the retrieved objects of this query are the nearest points around the query point.

An efficient nearest-neighbor search requires an indexing structure, which generates partitions for the feature vector space. Based on this indexing structure, only the objects in one or a few of partitions instead of in the whole feature vector space need to be visited during a nearest-neighbor search. Thus, the key issue of an indexing method is how to partition the feature space [11].

1.2 Boundary Problem

Researchers have developed many indexing methods for content-based retrieval in databases such as R-tree, R+-tree, R*-tree, SR-tree, Quad-tree, k-d tree, VP-tree, MVP-tree, and some other methods. However, these existing indexing methods usually confront the problem of performance degradation when the queries lie near the generated partition boundaries. The main reason of the problem is that these methods do not consider the natural clusters in the feature space.

The above problem is called as boundary problem. Generally, the objects in a multimedia database form some natural clusters in the corresponding feature vector space. Using nearest-neighbor search with a query, the retrieved objects almost belong to the same cluster which contains the query. However, the existing indexing methods often divide the objects in a natural cluster into several different nodes. As a result, when a query lies near partition boundary, the indexing methods will cause error if they only retrieve data from one partition, or will loss the efficiency if they have to visit many partitions for a search.

We build an indexing structure which partitions feature vector space based on the natural clusters to decrease the influence of the boundary problem. In this indexing structure, the partitioning boundaries approximatly are the natural clustering boundaries. The nearest-neighbor search on this indexing structure will become more efficient.

In the next section, we will introduce a noise resistance clustering algorithm, *Fuzzy Competitive Clustering (FCC)* and make a brief comparison between *FCC* and several traditional clustering algorithms. In Section 3, we will describe about how to build an efficient indexing structure, *FCC Binary Tree (FCC-b-tree)* by using *FCC*. Also, we will introduce two elimination rules for pruning the tree while searching and shown the experiment results in this section. Then we will come to our discussion and conclusion part in Section 4 and Section 5 respectively.

2 Fuzzy Clustering Methods for Indexing

We propose to use an efficient fuzzy clustering algorithm for content-based indexing in order to lessen the boundary problems mentioned in the pervious section.

2.1 Fuzzy Competitive Clustering (FCC)

Fuzzy Competitive Clustering (FCC) is an extension of traditional competitive learning. The main difference between FCC and traditional competitive learning is that in traditional competitive learning, the measurement in the competition step is the absolute distance, however, in FCC the measurement is the fuzzy membership value, which is a relative distance. It is more flexible and robust when compare with using absolute distance as a measurement.

The algorithm of FCC is outlined as follows.

(Step 0) Initialization: Every cluster in FCC is describe by fuzzy prototype [12, 13]. In the initialization, we randomly pick k points as the initial cluster prototype centers and every prototype have the same variance in each dimension as the initial variance of the cluster prototypes.

(Step 1) Competition: Calculate the fuzzy membership value for each data instance to each cluster prototype. The membership value u_{ik} for data instance x_k to cluster i is calculated from the equation:

$$u_{ik} = \frac{\sum_{j=1}^{a} u_{jik}}{a} , \qquad (1)$$

where a is the the number of attribute and u_{jik} is the membership value of data instance x_k to cluster i in j^{th} dimension.

u_{jik} can be any fuzzy membership function. In our experiment, we use *crisp* function as the fuzzy membership function and it is defined as:

$$u_{jik} = \frac{\sigma_{ji} + 1}{\sigma_{ji} + d(i,k) + 1} , \qquad (2)$$

where σ_{ji} is the variance of i^{th} cluster prototype in j^{th} dimension. $d(i,k)$ is the distance between instance x_k and the i^{th} cluster center.

After the claculation of fuzzy membership values, we increase the weighting, w_{ik}, of the k^{th} instance towards i^{th} cluster if the membership value of this data instance is the largest towards this cluster. The weighting is changed according to the following equation:

$$w_{ik} = \begin{cases} u_{ik} + \eta\Big(1 - u_{ik}\Big) & \text{if } u_{ik} \text{ is the largest,} \\ u_{ik} & \text{otherwise.} \end{cases} \qquad (3)$$

where, η, $0 \leq \eta \leq 1$, is the learning rate.

The changing process of w_{ik} is a simulation to the normalization process in traditional clustering algorithms. If u_{ik} is the the largest, then its weighting, w_{ik} should be the largest among w_{ik} for all k. Through this kind of *normalization like* process, we let each cluster prototype interact with each other to prevent generating *coincident clusters* [14].

(Step 2) Updating Cluster Fuzzy Prototypes: We update the cluster prototype by

$$m_i^{'} = \frac{\sum_{k=1}^{n} w_{ik}^2 x_k}{\sum_{k=1}^{k=n} w_{ik}^2} , \qquad (4)$$

$$\sigma_i^{'} = \frac{\sum_{k=1}^{n} w_{ik} \Big\| x_i - m_i \Big\|}{\sum_{k=1}^{k=n} w_{ik}} , \qquad (5)$$

where, $m_i^{'}$ is the new cluster centroid of i^{th} cluster and $\sigma_i^{'}$ is the new variance vector of i^{th} cluster. A variance vector stores the variance of each dimension for a cluster.

Steps 1 and 2 are iterated until the iteration converges or the number of iterations reaches a prespecified value. The final cluster prototypes are the results of the FCC.

2.2 Difference between FCC and traditional clustering algorithms

There are many different types of clustering algorithms, every method has its own advantages and disvantages, from K means clustering algorithm (KM), Competitive Learning (CL) to Fuzzy c means algorithm (FCM) [15]. However, almost all the common clustering algorithms [16, 17, 18, 19, 20, 21, 22] nowadays can divide into two groups. They are *probabilistic clustering* [23] and *possibilistic clustering* [24].

One of the great problem on Probabilistic Clustering algorithms is *Outliers*. Outliers are vectors, or called data point, in the data domain which are so distant from the rest of the other vector in the data set, that it would be unreasonable to assign them high membership values to any of the c clusters. Every Probabilistic Clustering algorithm obeys the constraint:

$$\sum_{i=1}^{c} u_{ik} = constant; \quad k = 1, ..., n , \qquad (6)$$

where, u_{ik} is the membership value of k^{th} instance to i^{th} cluster.

However, the above constraint does not permit all the c memberships to assume value lower than $1/c$. For an outlier x_k, all the ratios d_{ik}/d_{jk} will often be close to unity. This resulting that all the c membership values close to $1/c$. Because FCM and many other Probabilistic Clustering algorithms use the membership value as a weighting to calculate the cluster centroid. This unreasonable high membership values often cause improper positioning of the centroids. In fact, if an outlier is very distant, one of the centroids might position itself at the outlier's position.

On the other hand, possibilistic clustering also raise another problem. Use Possibilistic c means

(PCM) algorithm [25, 26] as an example. The objective function for PCM can break down into a sum of c single objective function. As a result, the centroids do not *affect* each other during the optimization process. This properties often leads to *coincident clusters*. Another problem for PCM is the result of PCM is heavily depends on initialization. The authors of [25] suggest to use FCM to initialize PCM. However, if an outlier is distanct, PCM will not able to recover from the *bad* initial partition generated by FCM.

However, FCC does not have the above problems. It is because in FCC, we do not have the constraint as in Eq. (6). Hence, we can assign small value of u_{ik} to an instance, if it is needed to do so. Also, as every cluster prototype interacts with each other, FCC would not generate *coincident clusters* like PCM. As we use fuzzy prototype to describe the cluster in FCC, FCC can be used to find the information of the cluster in each dimension.

Here we show a table to summarize the feature of these clustering algorithm:

Table 1: Comparison on the properities between *FCC* and several traditional clustering algorithms.

	CL	FCM	FCC	KM
Fuzzy	No	Yes	Yes	No
Inter-cluster data	Yes	Yes	Yes	Yes
Intra-cluster data	No	No	Yes	No
Noise-Resistant	Yes	No	Yes	No
$\sum_{i=1}^{c} u_{ik} = 1$	-	Yes	Not Needed	Yes
Knowledge on each dimension	No	No	Yes	No

2.3 Experimental Results

2.3.1 Experiments Setting

We test our method with synthetic data sets in the Gaussian Mixture distribution, whose probability density function can be written as follows:

$$p(\vec{x}) = \sum_{j=1}^{n} \alpha_j G(\vec{x}, \vec{m}_j, \Sigma_{\mathcal{X}_j}), \tag{7}$$

where n is the number of mixtures. Each weight, $\alpha_j \geq 0$ and $\sum_{j=1}^{n} \alpha_j = 1$, and each $G(\vec{x}, \vec{m}_j, \Sigma_{\mathcal{X}_j})$ is a single Gaussian function with the mean, $\vec{m}_j$ and the covariance matrix, $\Sigma_{\mathcal{X}_j}$.

In our experiments, we use the equal weight for each Gaussian mixture as follows,

$$\alpha_1 = \alpha_2 = \cdots = \alpha_n = \frac{1}{n}. \tag{8}$$

We also use a diagonal matrix as the covariance matrix of each Gaussian function.

$$\Sigma_{\mathcal{X}_i} = \begin{bmatrix} \sigma_i & 0 & \cdots & 0 \\ 0 & \sigma_i & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_i \end{bmatrix} \tag{9}$$

we set σ_i as a random variable with range from 1 to 10 and $n = 3$ for generating the testing data set.

2.3.2 Experiment Results

From the above experiment setting. We randomly initialize 10 different data set and apply several clustering algorithms on the data set to test their performance. The included clustering algorithms are, *fuzzy competitive clustering*, *competitive learning*, *rival penalized competitive learning*, *fuzzy c means clustering* and *K means clustering*. Each time after we finish the clustering, we use the clustering results to perform classification and compare the classification results with those given from *Maximum a posterior (MAP)* [27]. The results are summarized as follows:

Table 2: Comparison on the perfermance between *FCC* and several traditional clustering algorithms.

	Average Error Rate (in percentage)
Fuzzy Competitive Clustering	9.70
Fuzzy c Means	15.48
K Means	13.29
Competitive Learning	16.07
Rival Penalized Competitive Learning	13.12
Maximum a Posterior	5.59

3 FCC-b-tree: A Top-Down Hierarchical FCC Indexing Structure

3.1 Generating Top-Down Indexing Structure

After the FCC clustering is finished, the next step is to build an indexing structure. We realize the indexing structure using a top-down hierarchical clustering approach, which clusters the next level of feature vectors only based on the subset of data points partitioned in the previous level. Thus, the hierarchical clustering approach transforms a feature vector space into a sequence of nested partitions.

In this approach, the clusters at the same level do not overlap each other. Therefore, we can use a tree to describe the relationships between the different clustering levels. Especially, if we set a restriction that each set of data points in a level can only be partitioned into at most two subsets in the next clustering level, this tree becomes a binary tree. After clustering, all the feature vectors are in one cluster at the root level, level 0, and there are at most 2^i subtrees (clusters) at level i.

We can use the binary tree as indexing structure for nearest-neighbor search. The basic idea is that, at the root node of the binary tree, a query vector $\vec{q}$ is compared to the fuzzy prototypes of the clusters in the immediate lower level. The child node corresponding to the cluster with $\vec{q}$ having the highest membership value is selected. The elements in the selected cluster will be the result of the query if they satisfy the criteria of the nearest-neighbor search. Otherwise, the search will proceed to the lower levels.

Next, we outline the procedure of building a binary indexing tree using FCC clstering. We name the tree FCC-b-tree, FCC Binary Tree.

3.2 Building FCC-b-tree

Given a set of data, we perform top-down FCC clustering and build a FCC-b-tree based on the clusters. The basic idea is that we apply FCC to cluster the data set into two sub-clusters each time and then continue to do FCC clustering hierarchically to each of the sub-clusters until each of the final sub-clusters contains less than a pre-specified number of data points. With these FCC clusters, we can build a binary tree structure. There are two kinds of nodes in the tree: *leaf node* and *non-leaf node*.

A leaf node contains a cluster of at most M data points calculated by FCC clustering. M is the maximum number of data in a leaf node.

The algorithm for building the hierarchical binary tree by using FCC clustering is shown in the Appendix section.

Considering the update of FCC-b-tree, we next design two operations which insert or delete a single feature vector to or from the FCC-b-tree without re-clustering. Next, we show the two update operations.

3.3 Update of FCC-b-tree

3.3.1 Insertion

The algorithm for insering a single feature vector $\vec{p}$ to the FCC-b-tree is shown in the Appendix section.

The performance of the indexing tree for searching may be reduced after some individual data point insertions. The more the insertions, the worse the performance. The reason is that the insertion algorithm dose not fully consider the overall distribution of the inserted data point and the original data so that it cannot guarantee to keep the natural clusters. The searching performance will then be worse. As a result, we may have to rebuild the indexing structure after a certain amount of data points have been inserted.

3.3.2 Deletion

Apart from insertion, we can also delete an individual feature vector from a FCC-b-tree. The algorithm of deletion is shown in the Appendix section.

The deletion algorithm makes the searching performance worse. When the number of deletions increases, the searching performance will decrease because node merging will change the original indexing structure. Sometimes, the resultant indexing tree will give better searching results especially when the number of deletions is relatively

small. It is because only a few points are removed from the indexing tree and it does not affect the natural clusters and the indexing structure any more.

Next, we show how to realize retrieval using nearest nearest-neighbor search based on the FCC-b-tree indexing structure.

3.4 Nearest-neighbor Search Based on FCC-b-tree

The basic idea of our searching algorithm is that, given a query vector $\vec{q}$, we compare it to the fuzzy prototypes of the clusters in the immediate lower level. The child node corresponding to the cluster with $\vec{q}$ having the highest membership value is selected. Then, we test it with two *elimination rules*, to test whether or not the data instance in the node can be the nearest neighbor to the query. If it is possible, we check its child node too. Otherwise, we prune the sub-tree.

The elimination rules is based on the following properties of fuzzy prototype we used:

Property 3.1 (Membership and Distance) *Let, $D(c_i, x_k)$ is the distance between instance x_k and cluster i, u_{ik} is the fuzzy membership value for data instance x_k to cluster i.*

$$If \quad D(c_i, x_k) > D(c_i, x_m) \quad then \quad u_{ik} < u_{im} \quad (10)$$

Property 3.2 (Prototype of the query) *Given a query X, the variance in j^{th} dimension, $variance_j$ is equal for all j.*

Based on this property, we obtain the following elimination rules. Let X be the query, B be the point of current nearest neighbor of X among the features considered up to the present, S_p be the set of features associated with node p, N_p be the number of samples associated with node p, M_p be the sample mean of S_p, $u_{pmin} = \min_{u_{pk}} x_k \in S_p$, Min_p be the data instance in S_p with membership value u_{pmin}, B_p be the point on the line from M_p to X with $D(B_p, X) = D(B, X)$ and $u_{M_pB_p} \geq u_{M_pX}$.

Rule 3.1 (General Inclusion Rule) *$X_i \in S_p$ can be the nearest neighbor to X, if*

$$u_{XM_p} > u_{XB} \ .$$

Proof: If $u_{XM_p} > u_{XB}$, then by Property 3.1

$$D(M_p, X) < D(B, X) \ ,$$

it follows that,

$$\exists X_i \in S_p, D(X_i, X) < D(B, X) \ .$$

By the definition of *nearest search*, we directly conclude that, $X_i \in S_p$ can be the nearest neighbor to X.

Rule 3.2 (General Exclusion Rule) *No $X_i \in S_p$ can be the nearest neighbor to X, if*

$$u_{M_pB_p} < u_{pmin}$$

and

$$u_{XM_p} \leq u_{XB}$$

Proof: If

$$u_{XM_p} \leq u_{XB}$$

it follows that,

$$D(X, M_p) > D(X, B) \ .$$

And

$$D(X, M_p) \leq D(X, X_i) + D(X_i, M) \ ,$$

$$D(X, B_p) + D(B_p, M_p) \leq D(X, X_i) + D(X_i, M) \ ,$$

$$D(X, B_p) \leq D(X, X_i) + D(X_i, M) - D(B_p, M_p) \ ,$$

$$D(X, B_p) < D(X, X_i) + D(Min_p, M) - D(B_p, M_p) \ ,$$

because,

$$D(Min_p, M) - D(B_p, M_p) \leq 0 \ ,$$

then,

$$D(X, B_p) < D(X, X_i) \ ,$$

$$D(X, B) < D(X, X_i) \ ,$$

By the definition of *nearest search*, we conclude that, no $X_i \in S_p$ can be the nearest neighbor to X.

These rules is actually very similar to those in branch-and-bound algorithm [28] . The main different between these two rules and those in branch-and-bound algorithm is in branch-and-bound algorithm, they use the absolute distance as the measurement. While in our elimination rules, we use fuzzy membership value as the measurement, which is more flexible then using absolute distance as the measurement.

3.5 Efficiency Tests for FCC-b-tree

In this section, we are going to define the *efficiency* for the FCC-b-tree indexing method and demonstrate it through a set of experiments.

Definition 3.1 (Efficiency Measurement) *Let, x is the number of instances reached for the checked method, y is the number of instances reached in linear search, and z is the size of data.*

$$\begin{aligned} efficiency &= 1 - \frac{x}{y} \\ &= 1 - \frac{x}{z}. \end{aligned} \tag{11}$$

Then, we perform the experiments with the following setup and procedures.

3.5.1 Experiments Setting

We test our method with the same setting that described in Section(2.3.1), except we set the value of $n = 10$ and a total 10000 data instance for generating the testing data set.

3.5.2 Experiments Results

After a series of data retrieval, we calculate the maximum, average and minimum performance of the indexing structure. In calculating the average performance, we omit the the trials with the largest number of instance reached and the smallest number of instance reached. After performance calculation, we found the following:

Table 3: Efficiency measurement of FCC-b-tree.

Maximum Efficiency	0.87
Average Efficiency	0.79
Minimum Efficiency	0.59

4 Discussion

After introducing the FCC-b-tree for indexing and retrieval, we would like to have a short discussion on its performance.

1. **Efficient Nearest-neighbor Retrieval:**

   From most the experimental results, the efficiency is about 0.8 for 100%-accuracy nearest-neighbor retrieval in general. It is because FCC gives us natural clusters and we try to keep each of the natural clusters in a node. Therefore, most of the data in the node can be retrieved together as the result of a requested nearest-neighbor query and hence improve the effectiveness of retrieval. Besides, we make use of the elimination rules stated in Section 3.4 on the FCC-b-tree for retrieval so that the efficiency of nearest-neighbor search can be increased.

2. **Solving the Boundary Problem for 100% Nearest-neighbor Result:**

   By using the stopping criteria stated in Section 3.4 on the FCC-b-tree, the boundary problem described in Section 1 can be solved. For example, there is a nearest-neighbor query lying on a boundary of two cluster partitions. With the backtracking mechanism and the 2 elimination rules, the searching algorithm gives us 100% result of the query which may contain data objects on different clusters on both sides of the boundary efficiently.

3. **Insertion and Deletion:**

   FCC-b-tree has a hierarchical indexing structure which helps us to perform data insertion and deletion. From Sections 3.3.1 and 3.3.2, we know that a single data object can be inserted to or deleted from the indexing structure easily because there is a clear relationship among the internal nodes. Starting from the root node, we can find the target leaf node without any problem and then update the indexing structure for data insertion and deletion.

In summary, our method has good searching performance in general. The searching algorithm solves the boundary problem and makes nearest-neighbor search on the FCC-b-tree efficiently and effectively. Moreover, the hierarchical structure helps us to update the FCC-b-tree simply.

5 Conclusion

We have used an efficient clustering algorithm Fuzzy Competitive Clustering (FCC) to locate natural clusters for content-based indexing and retrieval. Based on the located clusters, we have proposed to build a hierarchical binary tree (FCC-b-tree) for retrieval. We also make use of some elimination rules to solve the boundary problem. From the experimental results, it is concluded that: (1) FCC gets the better cluster prototypes then other traditional clustering algorithms in general. (2)

The FCC-b-tree always has a better performance than linear search.

6 Appendix

Algorithm to build *(BuildTree(D, P, M))* the FCC-b-tree, insert *(Insert(T, p, M))* instance into and delete *(Delete(T, q, M))* instance from the FCC-b-tree:

Algorithm 1 BuildTree(D, P, M)

▷ *Input: A set of data objects D, a FCC-b-tree node P (P is empty at the first time), and the maximum node size M*

▷ *Output: A FCC-b-tree*

```
1  if D's size is greater than M then do
2      create a non-leaf node Q
3      add Q as a child node of P if any
4      use FCC to cluster D into two sub-sets D_1 and D_2
5      BuildTree(D_1, Q, M)
6      BuildTree(D_2, Q, M)
7      return Q
8  else
9      create a leaf node L for D
10     add L as a child node of P if any
11     calculate the clustering information of D and store it in the corresponding entry of P
12     return L
13 end if
```

Algorithm 2 Insert(T, p, M)

▷ *Input: A FCC-b-tree T, a to-be inserted data object p, and the maximum node size M*

▷ *Output: An updated FCC-b-tree*

```
1  N ← the root node of T
2  while N is not a leaf node do
3      N ← the node that p gives the highest membership value among its child nodes if any
4  end while
5  associate p to N
6  update the cluster prototype of N according to the FCC clustering rules
7  if N's size is larger than M then do
8      split the node into 2 sub-nodes by using FCC
9  end if
10 update the information of N's ancestors if necessary
```

Algorithm 3 Delete(T, q, M)

▷ *Input: A FCC-b-tree T, a to-be deleted data object q, and the maximum node size M*

▷ *Output: An updated FCC-b-tree*

```
1  N ← the root node of T
2  while N is not a leaf node do
3      N ← the node that p gives the highest membership value among its child nodes if any
4  end while
5  if q is associated with N then do
6      remove q form N
7      update the cluster prototype of N according to FCC clustering rules
8      update the information of Q's ancestors if necessary
9      if the size of Q's parent node less than M then do
10         merge all Q's parent node's child nodes
11     end if
12 end if
```

References

[1] I. King, A. Fu, L. Chan, and L. Xu, "Montage: An image database for the hong kong's textile, fashion, and clothing industry," 1995. http://www.cse.cuhk.edu.hk/~miplab.

[2] W. Niblack, R. Barber, W. Equitz, M. Flickner, E. Glasman, D. Petkovic, P. Yanker, C. Faloutsos, and G. Taubin, "The qbic project: querying images by content using color, texture, and shape," in *Proceedings of the SPIE - The International Society for Optical Engineering*, vol. 1908, pp. 173–187, 1993.

[3] A. Pentland, R. W. Picard, and S. Sclaroff, "Photobook: Content-based manipulation of image databases," *International Journal of Computer Vision*, vol. 18, pp. 223–254, June 1996.

[4] J. Smith and S. F. Chang, "Visualseek: a fully automated content-based image query system," in *ACM multimedia - international conference - 1996*, pp. 87–98, Nov. 1996.

[5] V. Ogle and M. Stonebraker, "Chabot: Retrieval from a relational database of images," *Computer*, vol. 28, pp. 40–48, Sept. 1995.

[6] M. H. O'Dochery, C. N. Daskalakis, P. J. Crowther, C. A. Goble, M. A. Ireton, J. Oakley, and C. S. Xydeas, "The design and implementation of a multimedia information system with automatic content retrieval," *Information Services and Use*, vol. 11, pp. 345–385, 1991.

[7] A. Gupta, T. Weymouth, and R.Jain, "Semantic queries with pictures: The vimsys model," in *Proc. 17th VLDB*, pp. 69–79, 1991.

[8] T. Kato, "Database architecture for content-based image retrieval," in *SPIE*, vol. 1662, pp. 112–123, 1992.

[9] W. W. Chu, A. F. Cardenas, and R. K. Taira, "Kmed: A knowledge-based multimedia medical distributed database system," *Information Systems*, vol. 20, no. 2, pp. 75–96, 1995.

[10] J. K. Wu, A. D. Narasimhalu, B. M. Mehtre, C. P. Lam, and Y. P. Gao, "Core: A content-based retrieval engine for multimedia information systems," *ACM Multimedia Systems*, vol. 3, pp. 25–41, Feb. 1995.

[11] I. King and T. K. Lau, "Comparison of several partitioning methods for information retrieval in image databases," in *Proceedings of the 1997 International Symposium on Multimedia Information Processing (ISMIP'97)*, pp. 215–220, 1997.

[12] J. Zhang and L. Zhang, "Learning fuzzy concept prototypes using genetic algorithms," in *Fuzzy Systems Conference Proceedings, 1999. FUZZ-IEEE '99. 1999 IEEE International*, vol. 3, pp. 1790–1795, 1999.

[13] E. Smith and D. Osherson, "Conceptual combination with prototype concepts," in *Cognitive Science 8(4)*, pp. 337–361, 1984.

[14] M. Barni, V. Cappellini, and A. Mecocci, "Comments on a possibilistic approach to clustering," in *IEEE Trans. Fuzzy System*, vol. 4, pp. 393–396, 1996.

[15] J. C. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms*. New York: Plenum Press, 1987.

[16] E. Backer, "Cluster analysis by optimal decomposition of induced fuzzy sets," in *Delft, The Netherlands: Delft University Press*, 1978.

[17] F. Sets and S., "Pattern recognition with fuzzy objective function algorithms," *Plenum Pres*, pp. 112–127, 1981.

[18] J. Dunn, "A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters," in *Journal of Cybernetics*, p. 3:32, 1974.

[19] J. Dunn, "Well separated clusters and optimal fuzzy-partitions," in *Journal of Cybernetics*, pp. 4:95–104, 1974.

[20] E. Ruspini, "A new approach to clustering," in *Information and Control*, pp. 15(1):22–32, July 1969.

[21] L. Zadeh, "Similarity relations and fuzzy orderings," in *Information Sciences*, vol. 3, pp. 177–200, 1970.

[22] S. Ovchinnikov, r fuzzy, and p fuzzy, "Fuzzy sets and systems," in *Similarity relations, fuzzy partitions, and fuzzy orderings*, pp. 40:107–126, 1991.

[23] R. Dav and E. Krishnapuram, "Robust clustering method: a unified view," in *IEEE Transactions on Fuzzy Systems*, vol. 5, pp. 270–293, 1997.

[24] R. Krishnapuram and J. Keller, "A possibilistic approach to clustering," in *IEEE Transactions on Fuzzy Systems*, vol. 1, pp. 98–110, May 1993.

[25] R. Krishnapuram and J. Keller, "A possibilistic approach to clustering," in *IEEE Transactions on Fuzzy System*, vol. 1, pp. 98–110, May 1993.

[26] J. Krishnapuram, R.; Keller, "The possibilistic c-means algorithm: insights and recommendations," in *Fuzzy Systems, IEEE Transaction*, vol. 4 3, pp. 385–393, 1996.

[27] C. Bouman and K. Sauer, "A generalized gaussian image model for edgepreserving map estimation," in *IEEE Transactions on Image Processing*, vol. 2, pp. 296–310, Jul. 1993.

[28] B. Kamgar and L. N. Kanal, "An improved branch and bound algorithm for computing k-nearest neighbors," *Pattern Recognition Letters*, vol. 3, pp. 7–12, Jan. 1985.