ROMANIAN JOURNAL OF INFORMATION SCIENCE AND TECHNOLOGY
Volume **14**, Number 1, 2011, 81–97

Clustering methods in data fragmentation[1]

Adrian Sergiu DARABANT[1], Laura DARABANT[2]

[1]Computer Science Department,
Babes Bolyai University, Cluj-Napoca, Romania
E-mail: `dadi@cs.ubbcluj.ro`

[2]Department of Electrical Engineering,
Technical University, Cluj-Napoca, Romania
E-mail: `ldarabant@et.utcluj.ro`

Abstract. This paper proposes an enhanced version for three clustering algorithms: *hierarchical, k-means and fuzzy c-means* applied in horizontal object oriented data fragmentation. The main application is focusing in distributed object oriented database (OODB) fragmentation, but the method applicability is not limited to this research area. The proposed algorithms produce fragments for an OODB database based on the analysis of inter-class relationships and user queries (applications) running on the system. Each class extension is clustered and the quality of resulting fragments is then evaluated and compared between the proposed algorithms and with results obtained from other object-oriented fragmentation techniques. Numerical experiments on different databases show an average improvement in query processing time of 17-30%. The test scenarios take in account different database sizes. The methods are applied to a small, medium and large database in order to verify their scalability.

Key-words: data clustering, object-oriented databases, fragmentation, distributed database.

1. Introduction

Database distribution as a concept has been introduced in order to cope with data locality when processing and to improve performance by parallelizing operations. Although a promising area, distributed databases did not gain exactly the

[1]This work is supported by the Romanian Ministry of Education in the frame of PN2 ID550/2007 and PD_611/2010.

expected development boom because of their inherent design complexity. Distribution is generally implemented in two steps: *fragmentation (horizontal and vertical)* and *allocation*. A good distribution design requires more information than a centralized database. One needs to account for data accesses and user queries in order to be able to fragment the data and then allocate the resulting fragments to the nodes of the system. Not all this information can be easily gathered or estimated before the actual data is already in the database - thus before the database actually exists. The analysis process by itself is not at all as straightforward as in a relational scenario because the database designer needs to input into equation two dynamical aspects:

- *the user applications that will run on the designed system*;
- *the interclass/object relationships.*

This work assumes that the applications are known *apriori* and focuses on automatically determining the relations between objects in order to cluster them in horizontal fragments according to their user applications access patterns. The proposed clustering algorithms take as input a numerical model of the database that can be automatically derived by analyzing the set of user applications. In the proposed numerical model each object is represented by an *N-dimensional* vector that quantifies:

- *the query behavior characteristics of the object;*
- *its relationships with other objects;*
- *its relationships derived from method calls.*

The vectors are then used as input by the clustering algorithms and the result is a set of clusters containing the associated objects grouped according to their similarity over the three aspects mentioned above.

The main contributions of this paper are: the improvement of the clustering methods for hierarchical and k-means methods in the case of complex interclass relationships, a new implementation for the fuzzy clustering method avoiding the suboptimal solution and a scalability proof/experiment of the proposed methods.

Fragmentation methods for OODB environments, or flat data models have been generally considered in Karlapalem [1], [4], [5], Ezeife [2]. In [5], authors add methods references to objects into the scene introducing thus a form of complex hierarchy. However no actual fragmentation methods are proposed, authors just suggesting using some modified existing algorithms as those in [8]. Gandeharizadeh proposes in [3] a method for object distribution over a multiprocessing parallel system by using an object dependency graph. Ravat [6] uses the Bond Energy Algorithm (BEA) for vertical and horizontal fragmentation in an object based model. The algorithm uses the predicate affinity in order to place objects into fragments. Ezeife [7] presents a set of algorithms for horizontally fragmenting OODBs with simple and complex data models. The ideas developed in [7] are using the algorithm developed for horizontal fragmentation in relational data models. Bellatreche et al. [9] propose a method that emphasizes the role of queries in the horizontal fragmentation in order to minimize the number of I/O operations in the database. Savonet proposes in [10] a method that represents the interclass relationships as a dependency graph as [3]. Its method tries to find a group of classes commonly used in a set of class methods for which the execution can be parallelized. Authors of this paper in [13,14] and Baiao in [11] address the fragmentation ordering problems. Works [13,14] propose and then improve a method for determining the impact of class fragmentation order while in [11] the problem is mentioned without a specific solution.

The authors proposed an initial clustering approach for object fragmentation on the context of simple attributes/methods with no inter-class relationships in [12] and then in [15].

2. Database numeric modeling

The first step is the database numerical representation. The query behavior characteristics for objects are obtained by collecting the significant conditional predicates from all queries and constructing the corresponding vector components. Given a *complex* hierarchy H, a *path expression* P is defined as $C_1.A_1. \ldots A_n$, n≥1 where: C_1 is an entry point in H, A_1 is an attribute of class C_1, A_i is an attribute of class C_i in H such that $C_i$is the domain of attribute A_{i-1} of class C_{i1}(1≤ i ≤ n). In the general case, A_i can be a method call. If i<n, then A_i must return a single complex type value (an object).

An *entry point* to the database is a reference to a known class that allows navigation from its instances to the rest of the connected database graph. Usually there are multiple entry points.

As presented in [12], a *query* is represented conceptually as a tuple with the following structure q=(Target class, Range source, Qualification clause), where:

– *Target class* – (query operand) specifies the root of the class hierarchy over which the query returns its object instances;

– *Range source* – a path expression starting from an entry point and specifying the source class hierarchy;

– *Qualification clause* – logical expression over the class attributes and/or class methods, in conjunctive normal form. The logical expression is constructed using atomic predicates: *path_expression θ value* where $\theta \in \{<, >, \leq, \geq, =, \neq, in, \supset, \supseteq\}$.

Let Q =$\{q_1, \ldots, q_t\}$ be the set of all queries running against the database to fragment. Let *Pred*=$\{p_1, \ldots, p_q\}$ be the set of all atomic predicates Q is defined on. Let *Pred(C)*=$\{p \in$*Pred*$|p$ ia a conditional on an attribute of class Cor on an attribute of its parent$\}$. Given the predicate $p \equiv C_1.A_1. \ldots A_n\ \theta\ value$, $p \in Pred(C_n)$ if class C_i is the complex domain of A_{i-1}, $i = 2...n$, and A_n has a complex type or simple type.

Given *two* classes C and C', where C' is subclass of C, $Pred(C') \supseteq Pred(C)$. Thus the set of predicates for class C' contains all the predicates directly imposed on attributes of C' and the predicates defined on attributes of its parent class C and inherited from it. This is a natural representation given that C' is a subclass of C it contains all of its attributes by inheritance.

Using the already defined notion we introduce the *object-condition matrix* for class C – *OCM(C)* – as *OCM(C)*=$\{a_{ij}, 1 \leq i \leq |Inst(C)|, 1 \leq j \leq |Pred(C)|\}$, where $Inst(C) = \{O_1, \ldots O_m\}$ is the set of all instances of class C, $Pred(C) = \{p_1, \ldots, p_n\}$. Similarly the *characteristic vector matrix* for class C is the set *CVM(C)*=$\{w_{ij}, 1 \leq i \leq |Inst(C)|, 1 \leq j \leq |Pred(C)|\}$.

$$OCM_{ij} = \begin{cases} 0, if\ p_j(O_i) & = false \\ 1, if\ p_j(O_i) & = true \end{cases}, \tag{1}$$

$$w_{ij} = CVM_{ij} = \frac{\sum\limits_{l=\overline{1..m}, a_{lj}=a_{ij}} [(a_{lj}|a_{lj} = 1) + (1 - a_{lj}|a_{lj} = 0)]}{m}.$$

Line i in *OCM(C)* is the *object condition vector* of O_i, where $O_i \in$*Inst(C)*. Object condition vectors represent the qualitative information about the way an object instance relates to a predicate in a user query. The CVM matrix is derived from OCM and gives the ratio of objects that behave in the same way for a given query predicate. Using the CVM or the OCM matrix depends on the type of the clustering algorithm. Some are working only with continuous vector components, others only with discrete values or with both continuous and discrete values. An example of the OCM and CVM matrices are given in Table 1.

We have captured so far all characteristics of simple attributes and methods. We need to express the class relationships in our vector space model. We first model the aggregation and association relations.

Given two classes C_O (owner) and C_M (member), where C_M is the domain of an attribute of C_O, a path expression traversing this link navigates from instances of C_O to one or more instances of C_M. We call *left derived fragmentation* the process where fragmentation is first performed on the owner class and then on the member class. Likewise, *right derived fragmentation* is the process where the member class is fragmented first, followed by the owner class. In the case of left derived fragmentation C_O will drive the fragmentation of C_M. In the right derived fragmentation variant the C_M will drive the fragmentation of C_O. Each of the two strategies is suitable for different query evaluation strategies. For example, in reverse traversal query evaluation strategy, the right derived fragmentation variant usually gives the best results. The query graph can be seen as a collection of path expressions all starting in its the *pseudo root*. In this context classes that are evaluated first are those fragmented first. They actually drive the fragmentation of the related classes and have less *entropy* (misplaced objects in the generated fragments). In forward traversal and left derived fragmentation the classes at the beginning of the path expression are fragmented first, yielding highly compact fragments. As we approach the end of along path expression the compactness degree of the generated fragments decreases. As a result the number of accessed fragments increases. We assume here, for space reasons, that right derived fragmentation method is used. However, both: the algorithms and the vector space model remain the same when considering left derived fragmentation order.

Table 1. The OCM and CVM matrices

OCM

Grad/Pred	Grade≤4	Dept.Name like "Info%"	Age()≥30	Sex=M	Sex=F	Grade≥8
G_1	0	1	0	1	0	1
G_2	1	1	1	1	0	0
G_3	0	0	0	0	1	1
G_4	1	0	1	1	0	0
G_5	0	0	1	1	0	0
G_6	0	0	0	0	1	1
G_7	1	0	0	0	1	0

CVM

Grad/Pred	Grade≤4	Dept.Name like "Info%"	Age()≥30	Sex=M	Sex=F	Grade≥8
G_1	0.38	0.53	0.69	0.69	0.69	0.31
G_2	0.61	0.53	0.31	0.69	0.69	0.69
G_3	0.38	0.46	0.69	0.31	0.31	0.31
G_4	0.61	0.46	0.31	0.69	0.69	0.69
G_5	0.38	0.46	0.31	0.69	0.69	0.69
G_6	0.38	0.46	0.69	0.31	0.31	0.31
G_7	0.61	0.46	0.69	0.31	0.31	0.69

Let $\{F_1, \ldots F_n\}$ be the fragments of C_M. Let $Agg(O_i, F_j)=\{O^m \mid O^m \in F_j, O_i \in Inst(C_O), O_i \text{ references } O^m\}$ be the set of objects referenced by O_i from the already formed clusters of class C_M. Given the set of fragments for C_M, the *attribute-link induced object-condition vectors for derived fragmentation* are defined as $ad_i = (ad_{i1}, ad_{i2}, \ldots, ad_{in})$, where each vector component is expressed by the following formula:

$$ad_{ij} = \operatorname{sgn}\left(|Agg(O_i, F_j)|\right), \quad j = \overline{1, n}\,. \tag{2}$$

For an object $O_i \in Inst(C_O)$ and a fragment F_j of C_M, ad_{ij} is 1 if O_i is linked to at least one object of F_j and is 0 otherwise.

Given the set of fragments for C_M, *attribute-link induced characteristic vectors for derived fragmentation* is defined as $wad_i = (wad_{i1}, wad_{i2}, \ldots, wad_{in})$, where each vector component is expressed by the following formula:

$$wad_{ij} = \frac{|\{O_l \in Inst(C_O) \mid \operatorname{sgn}(|Agg(O_l, F_j)|) = \operatorname{sgn}(|Agg(O_i, F_j)|)\}|}{|Inst(C_O)|}, \quad j = \overline{1, n}\,. \tag{3}$$

Each wad_{ij} component gives the percentage of objects in C_O that refers in the same way as O_i objects from F_j. Two objects O_i and O_l are said to link F_j *in the same way* if they are both either linked or not linked with objects from F_j. According to this criteria, two objects are candidates to be placed in the same fragment of C_O in respect to F_j if they are both related in the same way to F_j.

The following paragraphs present the class relationships induced by the presence of complex methods. Given a class with complex methods C(owner) that has to be fragmented, one should count in the fragmentation of classes referred by its complex methods. The method based relationships need to be introduced in the numerical representation in order to capture the method reference dependencies in the fragmentation process.

Let $MetComplex(C)$=$\{m_i \quad | \quad m_i$ *complex method of* $C\}$ – be the set of all complex methods of class C – methods referring instances of other classes.

Let $SetCRef(m,C)$=$\{C_R \quad | \; C \neq C_R, C_R$ *is referred by method* $m \in MetComplex(C)\}$ be the set of classes referred by the complex method m of class C. For a given instance of a class $C$with complex methods let:

$SetORef(m,O_i,C_R)$=$\{O_r^{'} \in Inst(C_R) \mid C_R \in SetCRef(m,C)$, $m \in MetComplex(C), O_r^{'}$ *is referred by method* m $\}$ – be the set of instances of class C_R, referred by the complex method m of class C.

For each pair $(m_k, C_R) \in \{m_k \in MetComplex(C)\} x SetCRef(m_k,C)$ we quantify the way each instance of C refers (through complex methods) instances from fragments of C_R. Given a class $C_R$referred by a complex method m_k of class C, and the fragments of class $C_R \longrightarrow \{\mathrm{F}_1, \ldots \mathrm{F}_n\}$, we define the *method-link induced object-condition vectors for derived fragmentation.* For each instance O_i of C let md_i= $(md_{i1}, md_{i2}, \ldots, md_{in})$ be the *method-link induced object-condition vector.* Each vector component is defined by the following formula:

$$md_{ij} = \operatorname{sgn}\left(|\{O_l \in Inst(C_R) | O_l \in F_j \cap SetORef(m_k,O_i,C_R)|\right) \;, \quad j=\overline{1,n}\,. \qquad (4)$$

Each md_{ij} evaluates to 1 when object $O_i \in Inst(C)$ refers objects from fragment F_j of class C_R and 0 otherwise. For each object O_i we obtain, for each pair (m_k, C_R), one *method-link induced object-condition vector.* We derive from it the *method-link induced characteristic vector for derived fragmentation,* wmd_j =$(wm_{i1}, wmd_{i2}, \ldots, wmd_{in})$, where:

$$wmd_{ij} = \frac{|\,\{O_l \in Inst(C) \mid md_{lj} = md_{ij}\}\,|}{Inst(C)} \;, \quad j=\overline{1,n}\;, \quad l=\overline{1,|Inst(C)|}\,. \qquad (5)$$

Each wmd_{ij} quantifies the way objects of class C refer objects from fragments of C_j through complex methods.

When modeling relationships induced by the presence of complex methods, we obtain as many referring condition vectors (object-condition and characteristic), for each instance O_i of C,as the number of elements of the Cartesian product $\{m_k \in Met\ Complex(C)\} \times SetCRef(m_k, C)$.

As the number of elements in $\{m_k \in MetComplex(C)\} \times SetCRef(m_k, C)$ is usually large we need to use some heuristics in order to retain only the pairs with significant impact in the fragmentation. In order for a pair (m_k, C_R) to be kept it should satisfy the following combined restrictions:

- The number of calls to the method m_k should be significant compared to the contribution brought by all method calls made by applications running on the database;

- The number of instances of CR referred by the method mk should be significant compared to the number of instances of all classes generally referred by the applications.

The above conditions are expressed equation (6) (*significance factor*). In (6) the first factor gives the ratio between the number of calls to method m_k and the number of calls of all complex methods of class C. The second factor gives the ratio between the number of C_R instances referred by m_k and the number of all objects referred by m_k.

$$Sig(m_k, C_R) = \frac{NrCalls(m_k)}{\sum\limits_{m_l \in MetComplex(C)} NrCalls(m_l)} \times$$

$$\times \frac{\sum\limits_{O_i \in Inst(C)} |SetORef(m_k, O_i, C_R)|}{\sum\limits_{C_p \in SetCRef(m_k)} \sum\limits_{O_r \in Inst(C)} SetORef(m_k, O_r, C_p)} \quad . \tag{6}$$

In reality the actual method parameters would normally influence the set of objects referred by the method. Even more, the set of referred objects could be as well influenced by the internal state of the object. However, tracking all the possible combinations is computationally intractable – even in simple situations. The statistical heuristic proposed in (6) is still manageable and helps reducing the problem space dimensions.

If the class C is related with classes $C_{A1}, C_{A2}, \ldots, C_{Ap}$ by means of complex attributes, and with classes $C_{M1}, C_{M2}, \ldots, C_{Mr}$ by means of complex methods, the *extended characteristic vector* we_i for object $O_i \in Inst(C)$ is obtained by appending the p attribute-link induced characteristic vectors and the number of $mc{=}|\{m_k{\in}MetComplex(C)\} \times SetCRef(m_k, C)|$ method-link characteristic vectors to the characteristic vector of O_i. However, as we have already mentioned above, we are using the *significance factor* to filter out non-relevant pairs (m_k, C_R) and vectors derived from them. A significance factor around 0.2–0.3 was used in the experiments. Values in this range filter out most of the inappropriate (m_k, C_R)pairs. Usually methods do not appear frequently in the queries so they have less impact than inter-object references.

The *extended object-condition vector* ae_i for an object O_i is obtained in the same way by appending its attribute-link and method-link induced object-condition vectors to its object-condition vector. We denote by $EOCM(C)$ and $ECVM(C)$ the extended object-condition and characteristic matrices for class C.

The aim of our method is to group into a cluster those objects that are similar to one another. Similarity between objects is computed using the Euclidian and Manhattan metrics:

$$sim_E(O_i, O_j) = 1 - \frac{d_E(we_i, we_j)}{|Inst(C)|}, \quad sim_M(O_i, O_j) = 1 - \frac{d_M(we_i, we_j)}{|Inst(C)|} \, . \tag{7}$$

We use sim_E and sim_M in (7) to measure the *closeness/similarity* of two objects. Both measures take values in [0,1]. We should note that all characteristic vectors have positive coordinates by definition.

3. Clustering Algorithms

The numerical model is ready and all object characteristics are now represented in the vector space induced over the database. The final step is to apply the clustering methods in order to determine and group the objects according to their common features. The following paragraphs present three clustering algorithm applied to data fragmentation.

3.1. Hierarchical Clustering

The first proposed algorithm is the hierarchical clustering algorithm. The characteristic of this algorithm is that it starts with each object in its own cluster and then unifies at each iteration the pairs of most similar clusters [16]. It builds a partial tree towards the root and stops when a cluster compactness criteria is reached.

Algorithm HierachicalAggFrag **is**
Input: Class C, Inst(C) to be fragmented, the similarity function *sim:Inst(C)*$\times$ *Inst(C)*$\rightarrow$*[0,1]*, m = |Inst(C)|, 1 <k $\leq$ m desired number of fragments, EOCM(C), ECVM(C).
Output: The set of hierarchical clusters F={F$_1$,..,F$_k$}.

```
Begin
For i=1 To Inst(C) do $F_i=\{O_i\}$;
$F=\{F_1,\ldots,F_m\}$;
While |F| >k do
$(F_{u*},F_{v*})$:=argmax$(F_u,F_v)[sim(F_u,F_v)]$;
$F_{new}=F_{u*}\cup F_{v*}$;
$F=F-\{F_{u*},F_{v*}\}\cup\{F_{new}\}$;
End While;
End.
```

Fig. 1. Algorithm **HierachicalAggFrag**.

EOCM and ECVM denote the *extended object condition* and *characteristic* matrices obtained as described in section 2. The similarity between two clusters is computed using the average inter cluster similarity:

$$sim(F_u, F_v) = \frac{\sum\limits_{a_i\in F_u}\sum\limits_{b_j\in F_v} sim(a_i, b_j)}{|F_u| \times |F_v|}. \tag{8}$$

3.2. k-Means clustering

The classical k-means algorithm takes the input parameter k and partitions a set of m objects into $k$clusters so that the resulting intra-cluster similarity is high but the inter-cluster similarity is low. Cluster similarity is measured in regard to the *mean* value of the objects in a cluster, which can be viewed as the cluster's *center of gravity* (*centroid*). First, the k-means algorithm randomly selects k of the objects, each of which initially represents a cluster mean or center. Each of the remaining objects is assigned to the cluster with highest similarity, based on the distance between the object and the cluster centroid. It then computes the new centroid for each cluster and redistributes all objects according to the new centroids. This process iterates until a criterion function converges. The criterion tries to make the resulting $k$clusters as compact and separate as possible.

Our version of the algorithm chooses as initial centroids the most representative objects by analyzing the input predicates and the object vectors. At each iteration, should an object have several potential destination clusters (same similarity with the centroid), the one with maximum *object to cluster* similarity is chosen. The *object to cluster similarity* is computed by using the *min* similarity with all objects from the cluster:

$$sim(O_i, F_c) = \min\{sim(O_i, a) | a \in F_c\} . \tag{9}$$

We also choose as criterion function the degree of compactness/homogeneity H of all clusters. For a given cluster F, this value is the difference between the maximum and minimum similarity of all pairs of objects in F:

$$H(F) = \max\{sim(a, b) \in FxF, a \neq b\} - \min\{sim(a, b) \in FxF, a \neq b\} . \tag{10}$$

```
Algorithm k-meansFrag is
Input: Class $C$, Inst($C$) to be fragmented, the similarity function
$sim: Inst(C) \times Inst(C) \rightarrow [0,1]$, m=|Inst($C$)|, $1<k \leq m$ desired number of fragments,
$OCM(C)$, $CVM(C)$, threshold_value.
Output: The set of clusters F={$F_1,\ldots,F_f$}, where $f \leq k$.
Begin
  Centr={$c_1,...,c_k$}=InitCentr(Inst($C$),$OCM(C)$,$CVM(C)$,k);
  F={$F_i | F_i=\{c_i\}$, $c_i \in$Centr, i=1..k}; $F' = \emptyset$;
  // initial object allocation to clusters
  For all objects $O_i$ do
    $F_{candidates}$={argmax$_{centr}$(sim($O_i$,$c_l$),l=1...k))};
    $F_{u*}$=argmax$_{sim}$(sim($O_i$,$f_c$),$f_c \in F_{candidates}$); $F_{u*}$= $F_{u*} \cup \{O_i\}$;
  End For;
  While $F'$ <>F and H(F)<threshold_value do
    For all $F_i \in$F recalculate centroid $c_i$;
    $F'$=F;
    For all objects $O_i$ do
      $F_{candidates}$={argmax$_{centr}$(sim($O_i$,$c_l$), l=1..k))};
      $F_{u*}$=argmax$_{sim}$(sim($O_i$,$F_c$),$F_c \in F_{candidates}$);
```

```
        $F'_v$=$F'_v$–{$O_i$}, where $O_i \in F'_v$;
        $F'_{u*}$= $F'_{u*}$∪{$O_i$};
        F′=F′–{$F'_l$| $F'_l = \emptyset$};
      End For;
    End While;
**End.**
**Function** InitCentr(Inst($C$),$OCM(C)$,$CVM(C)$,k) **is**
**Begin**
  Centr=∅; n=|Pred($C$)|;
  For i=1 to k do
    $c_i$=argmin[d($OCM(O_j)$,$u_i$)], $O_j \notin$Centr, i≤n;
    $c_i$=argmin(sim($O_j$,Centr)), $O_j \notin$Centr, i>n;
    If SUM(OCM($O_j$) < average(SUM($O_l$) discard($c_i$)
    Else
    Centr=Centr∪{$c_i$};
  End for;
  Return Centr;
**End Function**;
```

Fig. 2. Algorithm k-MeansFrag.

The initial centroids are not arbitrarily chosen. Using knowledge about the predicates initial centroids are semantically chosen to be as far as possible away one from each other so that resulting clusters have less probability to intersect and finally merging with each other, thus degenerating the solution. Centroids are also representing the most *active* predicates; i.e. that are satisfied by more than the average number of objects that are implied by any predicate from the user queries.

3.3. The fuzzy c-means clustering algorithm

Fuzzy c-means (FCM) is a method of clustering that allows one object to belong to multiple clusters, simulating thus a fine grained, object-level replication. The algorithm we propose for horizontal fragmentation is described in the following:

```
**Algorithm** Fuzzy-c-meansFrag **is**
**Input**: Class $C$, Inst($C$) to be fragmented, the distance function *dist:Inst(C)*× *Inst(C)*→*R*, m=|Inst(C)|, 1<k≤m desired number of fragments, *EOCM(C)*,
z the fuzziness factor, $\varepsilon$_prob the probability matrix change threshold, MaxSteps maximum number of iterations, MinMembershipProb minimum probability for an object to belong to a cluster, $\varepsilon$_centr threshold for centroid equality.
**Output:** The set of clusters F={$F_1$,...,$F_f$}, where f ≤ k.
**Var:** U=[$u_{ij}$] the probability matrix, i=1..m, j=1..k.
**Begin**
  InitializeRandomProbMatrix($U^{(0)}$)
      // *or* InitializeGuidedProbMatrix($EOCM(C)$,$U^{(0)}$,k);
f:=k; $F_j$:=∅, j=1..f;
```

```
p:=1;
Repeat
      For j:=1 to f do
```

$$c_j = \frac{\sum_{i=1}^{m} u_{ij}^{z} \cdot ae_i}{\sum_{i=1}^{m} u_{ij}^{z}}$$

```
    End For;
    ReduceIdenticalClusters(Centr$^{(p)}$, U$^{(p)}$, f, $\varepsilon$_centr);
    // update $U^{(p)}$ using $U^{(p-1)}$
    For i:=1 to m do
      For j:=1 to f do
```

$$u_{ij} = \frac{1}{\sum_{l=1}^{f} \left(\frac{dist(ae_i, c_j)}{dist(ae_i, c_l)} \right)^{\frac{2}{z-1}}}$$

```
      End For;
    End For;
  Until (max{ |u$_{ij}^{(p)}$- u$_{ij}^{(p-1)}$| } $\leq \varepsilon$_prob) or p $\geq$ MaxSteps
  For i:=1 to m do
    For j:=1 to f do
      If u$_{ij}^{(p)}$ $\geq$ MinMembershipProb then F$_j$ = F$_j \cup$ {O$_i$};
      End For;
    End For;
  End.
```

```
Procedure InitializeRandomProbMatrix(U)
  For i:=1 to m do
    For j:=1 to f do
      u$_{ij}$:=random(0..1);
    End For;
  End For;
  // standardize cluster membership probabilities for each object
  // so that sum(u$_{ij}$, j=1..f)=1, for each object O$_i$, i=1..m.
  For i:=1 to m
    sum:=0;
    For j:=1 to f do sum:=sum + u$_{ij}$; End For;
    For j:=1 to f do u$_{ij}$:=u$_{ij}$/sum; End For;
  End For;
End Procedure;
```

```
Procedure InitializeGuidedProbMatrix($EOCM(C)$,U, k)
  Centr:={O$_j$, an object from Inst(C)};
  For j:=1 to k do
    $c_j$:=argmax(dist($O_j$,Centr)), $O_j \notin$Centr);
    Centr:=Centr$\cup\{c_j\}$;
  End for;
  For i:=1 to m do
    If $O_i \in$Centr then
      l:=indexOf($O_i$,Centr);
      $u_{il}$:=1;
      For j:=1 to k, j$\neq$l do
        $u_{ij}$:=0;
      End For;
    Else
      l:=argmin(dist($O_i$,Centr)); // the most similar centroid with $O_i$
      $u_{il}$:=1;
      For j:=1 to k, j$\neq$l do
        $u_{ij}$:=0;
      End For;
    End If;
  End For;
**End Procedure**;

**Procedure** ReduceIdenticalClusters(Centr, U, f, $\varepsilon$_centr)
  i:=1;
  While ( i<f ) do
    j:=i+1;
    While ( j$\leq$f ) do
      // *component wise equality*
      If max$\{|c_j^{(k)} - c_i^{(k)}|$, k=1..length($c_i$)$\}< \varepsilon$_centr then
        // *merge cluster j into cluster i and adjust*
        // *object membership probabilities for the unified cluster*
        $u_{li}$ := $u_{li}$ + $u_{lj}$, l=1..m;
        f := f–1;
      Else
        j := j+1;
      End If;
    End While;
  End While;
**End Procedure;**
```

Fig. 3. Algorithm fuzzy c-MeansFrag.

The algorithm starts with an initial probability matrix $U^{(0)}$. An element u_{ij} of this matrix expresses the membership probability of object O_i to the cluster F_j. The sum of membership probabilities for an object to all clusters must be equal to one. The probability matrix is optimized in an iterative manner. Each iteration starts by determining the centers of each fuzzy cluster, *Centr*=$\{c_1,\ldots, c_f\}$ in the algorithm.

Next we adjust the membership probabilities to the clusters represented by the new centroids – line (a2) in the algorithm. This iterative process aims to minimize the following objective function:

$$J_z = \sum_{i=1}^{m} \sum_{j=1}^{f} u_{ij}^z \cdot dist(ea_i, c_j) \text{ where } ea_i \text{ is the object vec} \tag{11}$$

The u_{ij} and c_i are updated in such way that the objective function be minimized in each iteration step. The iterative process stops when changes in the probability matrix between two consecutive steps is bellow a threshold value, ε_prob. The J function generally has saddle points and the generated probability series u_{ij} are not always convergent. In order to ensure optimal results the algorithm is applied multiple times with different initial centroids choice.

At the end of the iterative process we build the horizontal fragments for the class C by assigning to each fragment those objects for which the membership probability to that fragment exceeds a threshold value, *MinMembershipProb.* We chose as threshold value $1/f$ (the average membership probability of an object to all clusters).

In the iterative process, two centroids might become *equal.* Two vectors v_1 and v_2 are equal in our case, if v_2 is located in an ε_centr-neighborhood of v_1, or vice versa. If there are equal centroids this means that the corresponding clusters would contain approximately the same objects or objects with the same properties (they respect predicates and relationships in the same way). A set of clusters with equal centroids are called *degenerated clusters.* As they do not have a distinct semantic for the fragmentation we merge all degenerated clusters with equal centroids. The resulting cluster will accumulate all objects of the source degenerated clusters by summing their membership probabilities. Degenerated clusters may appear when requesting more fragments than can be separated in our modeled vector space.

4. Results and Evaluation

In this section we illustrate the experimental results obtained by applying our fragmentation scheme on experimental databases under different scenarios: *small, medium* and *large datasets.* As order of magnitude, a small dataset contains a few hundreds to a thousand class instances (for a given class). A medium dataset contains around a few thousands to tens of thousands instances and a large dataset contains from tens of thousands to hundreds of thousands to a millions instances (for a given class). Given a set of queries, we first obtain the horizontal fragments for the classes in the database; afterwards we evaluate the quality and performance of the fragmentation results by allocating fragments to the nodes of a virtual system. Each fragment is allocated to the node where is most used. Finally we use these to determine the impact of the resulted fragments when queries are run against the database. In our experiments we used a varying number of queries (between 10–50 queries).

For measuring the fragmentation quality we determine the cost of query remote accesses combined with the cost of query local irrelevant accesses to each fragment.

Remote accesses are made by applications running on a given node and accessing objects that are not stored on that node. Local irrelevant accesses are given by local processing incurred when a query accesses a fragment. Each access to a fragment implies a scan to determine objects that satisfy a condition. Irrelevant local access measure the number of local accesses to objects that will not be returned by the query. Intuitively, we want that each fragment be as compact as possible and contain as much as possible only objects accessed by queries running on the fragment's node. Objects in a fragment not accessed by any query running on it are just a burden and potential candidates to be placed in different fragments.

We use the following measure for calculating the fragmentation quality:

$$FPE(C) = FEM + FER \tag{12}$$

$$FEM(C) = \sum_{t=1}^{T}\sum_{i=1}^{M} freq_{ts} * \left| F_i - \left(Acc_{C\,t} - \bigcup_{\substack{j=1 \\ F_j \in s}}^{i-1} F_j \right) \right| \tag{13}$$

$$FER(C) = \sum_{t=1}^{T}\sum_{s=1}^{S} freq_{ts} \cdot \left| Acc_{C\,t} - \bigcup_{\substack{i=1 \\ F_i \in s}}^{M} F_i \right| \cdot \left| FragCover\left(Acc_{C\,t} - \bigcup_{\substack{i=1 \\ F_i \in s}}^{M} F_i \right) \right| \tag{14}$$

$Acc_{C,t}$ represents the set of objects accessed by query t from class C. $freq_{ts}$ is the frequency of query t running on site s. In (14) $s$is the site where F_i is located. M is the number of fragments for class C, T is the number of queries and S is the number of sites. The *FEM* term calculates the local irrelevant access cost for all fragments of a class. For a fragment and a query the irrelevant objects are those that:

a) are not accessed by the query or

b) are accessed by the query but are replicas of objects from other fragments already considered in the evaluation.

This comes from the natural fact that a query will only refer one replica of a given object when evaluated. *FER* calculates the remote relevant access cost for all fragments of a class. The second factor expresses the number of remote accessed objects, for a given query $t$running on a site s. For a given query running on a given node, *FragCover* calculates an optimal covering scheme of the set of remote accessed objects, formed only with remote fragments, so that only one replica of each remote object is considered.

The evaluation was conducted in each scenario on a centralized version of the database, a fully replicated one and the fragmented database obtained with the above algorithms. The fragments are allocated over a network of four to sixteen similar architecture and processing power computers. The fully replicated database contains a copy of all data on each node together with the processing logic. The result where also compared to the method presented by Ezeife and Barker in [7].

Figure 4 shows the detailed FPE values for the small database scenario. The three algorithms are represented with both similarity measures (Euclid and Manhattan). The last tree entries are the FPE values for the Min_Complete implementation [7] and a centralized and fully replicated database. The Fuzzy algorithm obtains an almost 30% gain as opposed to the other methods. The worst performing is the centralized database due to its low parallelizing performances and local irrelevant access cost.

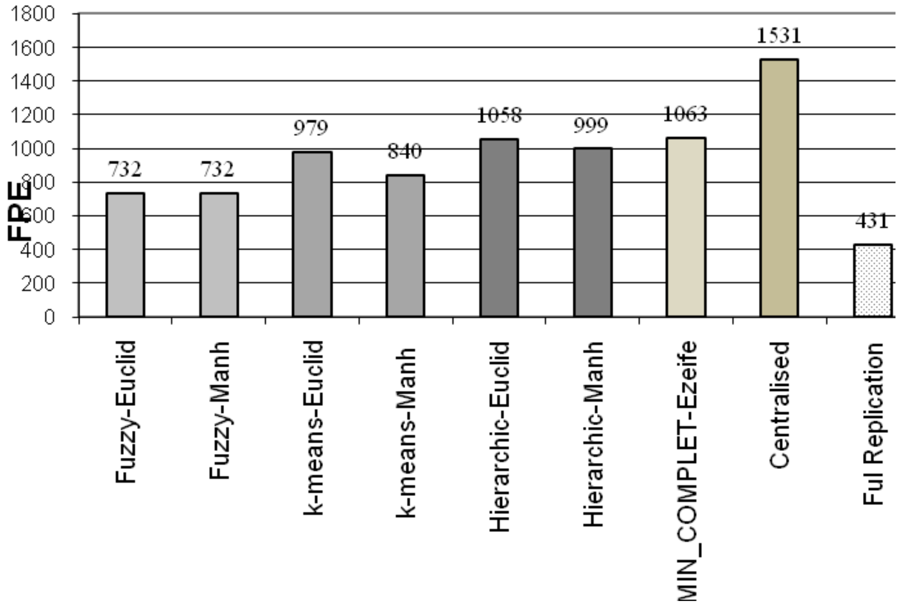

Fig. 4. Comparative FPE clustering results.

The fully replicated database has the best results. However, the conducted tests do not consider update operations and the FPE measure does not reflect their high cost for replicas.

Figure 5 deals with scalability. It shows the results of applying clustering on small, medium and large databases. Each level here is an order of magnitude larger the previous one in number of records. They are setup as tens of thousands, hundreds of thousands and millions of records. As seen the fuzzy method keeps its best place with the smaller cost (best quality fragments) because of the fine grained replication achieved. The k-means and min_complete algorithm behave similarly. The fragment quality for k-means is slightly better than min_complete in all cases. The hierarchical clustering does not scale well as unifying entire clusters always introduces outliers that cannot be corrected in the following steps. While on the same line with the other methods for small databases its costs explode for the medium and large database scenarios.

Figure 5 presents only the best costs for a method in the respective scenario, the best result throughout similarity measure choices and algorithm variations - for each distinct algorithm. Although the fuzzy clustering method seams to achieve improvements of around 30%, in the medium and mostly the large scenario it still has a hidden cost. In order to obtain the best result and achieve the optimal solution the algorithm should be executed multiple times with restart points when not converging properly. This gives a running time that is several degrees higher than for the other methods.

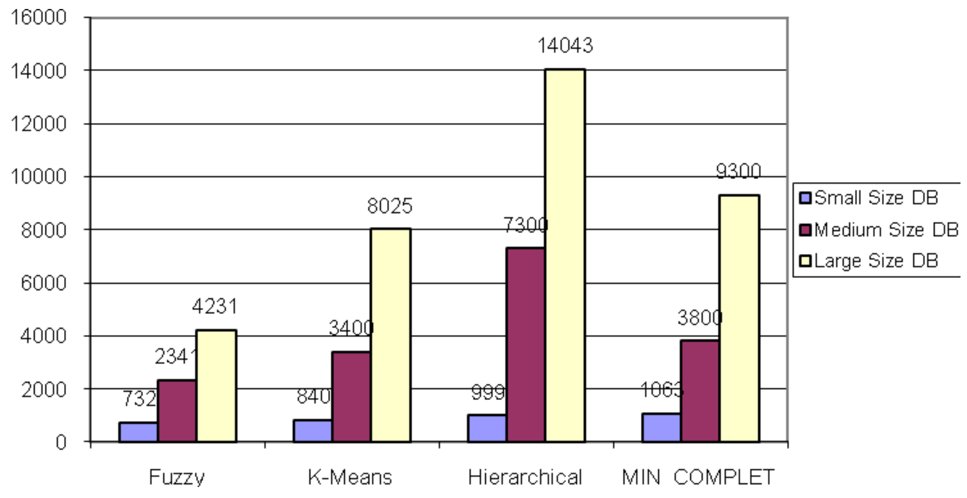

Fig. 5. Comparative results for small, medium and large datasets.

5. Conclusions

The paper presents three variations of some clustering algorithms with applications in database fragmentation. The proposed improvements seem to allow achieving better results in the case of fuzzy c-means and k-means algorithms. These are proven by the experimental results. Beside the hierarchical clustering, the fuzzy c-means and k-means methods are scalable enough to be applied to large datasets. The presented algorithms allow for algorithmic data fragmentation alleviating the need for a manual analyze of deep inter-class relationships by the database administrator.

References

[1] Karlapalem K., Navathe S. B., Morsi M. M. A., *Issues in distribution design of object-oriented databases*, in: Tamer Ozsu, M., Dayal, U., Valduriez, P. (eds.): *Distributed Object Management*, Morgan Kaufmann Publishers, pp. 148–164, 1994.

[2] Ezeife C.I., Barker K., *A Comprehensive Approach to Horizontal Class Fragmentation in a Distributed Object Based System*, International Journal of Distributed and Parallel Databases, **3**(3), pp. 247–272, 1995.

[3] Ghandeharizadeh S., Wilhite D., P*lacement of Objects in Parallel Object-Based Systems*, Technical Report 94-589, Department of Computer Science – University of Southern California, 1994.

[4] Karlapalem K., Li Q., *Partitioning Schemes for Object-Oriented Databases*, *Proceedings of the Fifth International Workshop on Research Issues in Data Engineering-Distributed Object Management*, Taiwan, pp. 42–49, 1995.

[5] Karlapalem K., Li Q., Vieweg S., *Method Induced Partitioning Schemes in Object-Oriented Databases*, *Proceedings of the 16th Int. Conf. on Distributed Computing System (ICDCS'96)*, Hong Kong, pp. 377–384, 1996.

[6] Ravat S., *La fragmentation d'un schema conceptuel oriente objet*, *Ingenierie des systemes d'informaton (ISI)*, **4**(2), pp. 161–193, 1996.

[7] Ezeife C. I., Barker K., *Horizontal Class Fragmentation for Advanced-Object Modes in a Distributed Object-Based System*, *Proceedings of the 9th International Symposium on Computer and Information Sciences*, Antalya, Turkey, pp. 25–32, 1994.

[8] Navathe S. B., Ra M., *Vertical Partitioning for Database Design: A Graphical Algorithm*, *Proceedings of ACMSIGMOD International Conference on Management of Data*, pp. 440–450, Portland, Oregon, 1989.

[9] Bellatreche L., Karlapalem K., Simonet A., *Horizontal Class Partitioning in Object-Oriented Databases*, Lecture Notes in Computer Science, vol. **1308**, Toulouse, France, pp. 58–67, 1997.

[10] Savonnet, M. et al., *Using Structural Schema Information as Heuristics for Horizontal Fragmentation of Object Classes in Distributed OODB*, *Proc. IX Int. Conf. on Parallel and Distributed Computing Systems*, France, pp. 732–737, 1996.

[11] Baiao F., Mattoso M., *A Mixed Fragmentation Algorithm for Distributed Object Oriented Databases*, *Proc. of the 9th Int. Conf. on Computing Information*, Canada, pp. 141–148, 1998.

[12] Darabant A. S., Campan A., *Semi-supervised learning techniques: k-means clustering in OODB Fragmentation*, IEEE International Conference on Computational Cybernetics ICCC 2004, Vienna University of Technology, Austria, August 30 – September 1, pp. 333-338, 2004.

[13] Darabant A. S., Campan A., *Optimal Class Fragmentation Ordering in Object Oriented Databases*, Studia Universitatis Babes Bolyai Informatica, vol. **XLIX**, no. 1, pp. 45–54, 2004.

[14] Darabant A. S., Campan A., *A new approach for optimal fragmentation order in distributed object oriented databases*, Pure Mathematics and Applications, vol. **15**(2–3), pp. 113–126, 2004.

[15] Darabant A. S., Campan A., Cret O., *Using Fuzzy Clustering for Advanced OODB Horizontal Fragmentation with Fine-Grained Replication*, Databases and Applications - DBA 2005, pp. 116–121, 2005.

[16] Darabant A. S., Campan A., Cret O., *Hierarchical Clustering in Object Oriented Data Models with Complex Class Relationships*, *Proceedings of the 8th IEEE International Conference on Intelligent Engineering Systems*, Cluj-Napoca, pp. 307–312, 2004.