Clustering of Composite Objects for CAD Databases

Hyun Syug Kang, Yong Ju Hwang
Dept. of Computer Science
Chonbuk National University
Chonju, Chonbuk 560-756, Korea

Chi Su Wu
Dept. of Computer Science and Statistics
Seoul National University
Kawnak Gu, Seoul 151-742, Korea

Abstract

Composite objects are descriptions of collections of components which themselves can be collections of their subcomponents. Efficient storage structuring and fast retrieval of the composite objects are essential to computer aided design applications of database management systems. This paper introduces two clustering methods: a structural clustering and a function-propagation clustering, in which all components of a category are clustered into an equivalent object cluster. The structural clustering is an extension of the component aggregation method and the function-propagation clustering is an extension of the functional clustering method. These two clustering techniques partition the database into meaningful and application-oriented clusters.

Several clustering methods are compared to demonstrate the performance enhancement when composite objects are organized according to our clustering methods.

1. Introduction

Composite objects are represented as collections of heterogeneous object structures retrieved together. The efficient organization and the fast retrieval of the composite objects are the major problems in design database application environment. Concerning this, much attention has been paid [KIM88, STON87, DITT88]. Especially, design database management system must have fast data retrieval strategy in displaying a composite object, since display is the operation most frequently used in the design database applications. To implement this operation, it is necessary to access all of the components constituting a nonprimitive composite object. For that reason, it is significant to store the components close to each other.

Recently many clustering methods [BANE88, CHEI88, HAFE88, KETA88] are proposed to manage these composite objects. Some of them apply only to class objects [HAFE88, KETA88], and others apply only to instance objects [BANE88, KIM88]. But, they are not combined with each other naturally. To cluster composite objects in object-oriented systems efficiently, it is necessary to combine two types of clustering methods.

To combine two type clustering methods effectively, we propose two clustering methods: structural clustering method, which is based on a partition rearrangement technique, and function-propagation clustering method, which is based on a cluster propagation technique. The former is an extension of the component aggregation method of Ketabchi and Berzins [KETA88], and the latter is an extension of the functional clustering method of Cheiney and Kieman [CHEI88].

In the component aggregation method, objects having exactly the same types of components are considered equivalent. The sets of objects equivalent by the method form component categories. This method allows efficient execution of frequent access patterns in engineering application. The primary objective of the method is to partition the database so that each partition contains data elements retrieved together, and to reduce the amount of irrelevant partitions accessed.

By extending the component aggregation method, we propose a new clustering method called the structural clustering. The method is based on the partition rearrangement technique, whose basic idea is that the partitions are categorized into several groups, and each group in which partitions are relevant to each other, may be clustered. Because two partitions closely related are referred to each other more frequently than unrelated ones, we store them close to each other.

Both of the component aggregation method and the partition rearrangement technique employ the structural relationship (part-of}
relationships among class (or type) objects. However, each class object consists of instance objects which have the same properties.

In the functional clustering method, all tuples whose function values are equivalent to each other are stored together. The method is created on a relational database system. We also propose another new clustering method called the function-propagation clustering, which is based on the method but applied on composite objects.

A composite instance object consists of different typed instance objects. It is possible to classify all instance objects of a class object into several categories according to their function values. And also these categories are to be propagated to their component objects. We call it the cluster propagation. The function-propagation clustering is based on this concept.

In this paper, Chapter 2 discusses the previous works on the clustering of composite objects. Chapter 3 provides a brief review of the component aggregation method and develops the structural clustering method by extending it. Chapter 4 discusses the function-propagation clustering method and the cluster propagation technique. Chapter 5 compares several clustering methods to demonstrate some benefits of our clustering methods. Chapter 6 offers concluding remarks and directions for future work.

2. Related Works

There are several clustering methods which exploit the structural relationships among composite objects.

To manage composite objects, Kim et al. [KIM87] suggests a clustering method based on the component reference graph among composite objects. In this method, whenever a higher level composite object is referenced, the lower level component objects are ready to be used automatically according to the component reference graph. However, since the lower level component objects are not always stored close to the higher level composite object, the access time of a composite object and its components together from the secondary storage is still slow.

Hafez et al. [HAFE88] designs the partial normalized storage model of nested relations. The model reorganizes all nested relations and stores them into a "near" optimum storage structure by using the relationships among composite objects. However, since this method stores a composite object as a tree structure, it does not support the direct acyclic graph structure of it.

To manage DAGs of composite objects, Benerjee et al. [BANE88] uses three traversal methods: the well-known depth-first and breadth-first traversals, and a children depth-first traversal. They organize the nodes of DAG into a clustered sequence of nodes, which is the sequence in which the nodes are visited in a given traversal method. This method is adequate in case of requiring all components of a composite object. However, since the instance objects of the same class object are distributed to each cluster sequences, it is far from being satisfactory in the kinds of queries applied to the same typed instance objects.

Ketabchi and Berrins [KETA88] suggest the component aggregation method, in which all component objects of composite objects are classified into several component classes and all component objects of each component class are placed in each partition. The method works well in managing the composite objects highly shared to many superobjects. The structural clustering method we employed is an extension of the method. We will describe it in Chapter 3.

There are also several clustering methods which exploit the semantic relationships among attribute values of the composite objects.

In a relational database management system SABRINA, Cheiney and Kieman [CHEI88] propose the functional clustering method, in which all tuples having the equivalent function values are stored together in the secondary storage. Those functions used as clustering predicates are most often used in the restriction part of queries. Therefore, they optimize the retrieval of tuples that would otherwise require processing the whole relation. Function-propagation clustering method we adopted is based on this functional clustering method but applied on composite objects. We will describe it in Chapter 4.

Dittrich and Lorie [DIT188] suggest the logical version cluster method, in which all versions of a class object are classified into version clusters by their semantics and they are stored according to their version clusters. This method allows for the meaningful grouping of versions, thereby efficient controlling of them.

3. Structural Clustering

Structural clustering is a method in which all related component objects are stored together according to the structural relationship among composite objects. This method is an extension of the component aggregation concept in the mathematical model proposed by Ketabchi and Berzins [KETA88].

3.1 Component Aggregation Concept

Object partitioning method of the mathematical model uses the component aggregation concept. Refer to [KETA88] for more detailed description of it. In this chapter, the basic concept of it and several new notations to be used afterwards are described. First, let's see an example depicted in figure 1.
The component referential relationship graph represents some composite objects. For example, in figure 1, the composite object M is composed of two component objects P and O, and the composite object P in turn composed of two component objects R and S, and so on. From this graph, it is also possible to develop a direct component function Pa(). The following definition illustrates this point.

[Definition 1] For the set of all objects, Ω, the direct component function of an element x, Pa(x) is defined by

\[ Pa(x) = \{ y \in \Omega \mid y \in x \} \]

where, notation \( \in \) (or \( \subseteq \)) represents a direct reference from a superobject to a subobject. For example, C \( \in \) T in figure 1 represents that the subobject C is directly referenced by the superobject T. The followings are the direct component functions corresponding to figure 1.

- \( Pa(A) = \emptyset \)
- \( Pa(B) = \emptyset \)
- \( Pa(C) = \emptyset \)
- \( Pa(R) = \{ A \} \)
- \( Pa(S) = \{ A, C \} \)
- \( Pa(T) = \{ B, C \} \)
- \( Pa(O) = \{ R, S \} \)
- \( Pa(P) = \{ A, R, S \} \)
- \( Pa(Q) = \{ R, T \} \)
- \( Pa(N) = \{ O \} \)
- \( Pa(M) = \{ R, P \} \)

There are several objects which have no component object such as A, B, and C. Specially, we define them as follows.

[Definition 2] The set of basic component objects, BP is defined by

\[ BP = \{ x \in \Omega \mid Pa(x) = \emptyset \} \]

For example, \( BP = \{ A, B, C \} \) in figure 1.

Based on the basic component objects, basic component representation can be defined subsequently.

[Definition 3] Basic component representation of an object x, \( P(x) \) is defined by

\[ P(x) = \{ y \in BP \mid y \in x \} \]

where, notation \( \in \) is the component referential relationship representing a direct reference or an indirect reference. Notice that the relationship \( \in \) has the transitive property. For example, in figure 1, since \( O \in S \) and \( S \in A \), \( O \in A \) exists. Relationship \( \in \) also contains relationship \( \subseteq \) which is a reference from itself. For example, in figure 1, \( P(O) = \{ A, C \} \) and \( P(A) = \{ A \} \).

We can also define the component classes with the basic components.

[Definition 4] The set of component classes of all objects, BT is defined by

\[ BT = \{ (X, P^{-1}(X)) \mid X \subseteq BP \} \]

where, \( P^{-1}(X) = \{ y \in \Omega \mid \exists x \in X \setminus P(y) = x \} \).

Next, let the component class of an object x be \( <x> \). Then, in figure 1, we can classify all composite objects into component classes. The followings are the result of the classification.

- \( <A> = (\{ A \}, \{ A, R \}) \)
- \( <B> = (\{ B \}, \{ B \}) \)
- \( <C> = (\{ C \}, \{ C \}) \)
- \( <S> = (\{ P \}, \{ P \}) \)
- \( <M> = (\{ A \}, \{ C \}, \{ S \}, \{ P \}, \{ O \}, \{ M \}) \)
- \( <Q> = (\{ N \}, \{ Q \}, \{ N \}) \)
- \( <T> = (\{ B \}, \{ C \}, \{ T \}) \)

Each component class is represented as a pair of sets. The front set represents the corresponding basic component objects, and the rear set represents the composite objects in the component class. Hence we can allocate each partition to each component class as follows.

- Partition 1 (P1) : \{N, Q\}
- Partition 2 (P2) : \{M, P, O, S\}
- Partition 3 (P3) : \{T\}
- Partition 4 (P4) : \{R, A\}
- Partition 5 (P5) : \{B\}
- Partition 6 (P6) : \{C\}

According to this partitioning convention, all composite objects in a partition are stored together (for example, they are stored in a block or neighbor blocks of disk).

As is well known in [HEIL87], a composite object and its components are usually treated together. The component classes are partitioned according to the characteristics of composite objects, and the component objects in the same component class are stored in the same partition. Since the component aggregation method enables us to access composite objects without accessing unrelated objects, it is an efficient clustering method for managing the composite objects in the computer aided design environments.

3.2 Partition Rearrangement
We suggest, in this subchapter, an effective clustering method, the structural clustering method which is the extension of the previous component aggregation concept. This method uses the referential relationships among objects in different partitions. At first, let's define them.

[Definition 5] The referential relationship from a partition $P_i$ to a partition $P_j$, $P_i \Rightarrow P_j$ is defined by

$$P_i \Rightarrow P_j : \exists x, y \in P_i \wedge y \in P_j \wedge x \nless y.$$

The structural clustering is created by using these referential relationships among partitions. The referential relationships are used to calculate the partition referential weight values as shown in [Definition 6].

[Definition 6] If the referential relationship from a partition $P_i$ to a partition $P_j$, $P_i \Rightarrow P_j$ exists, the partition referential weight value from the partition $P_i$ to the partition $P_j$, $PRWV(P_i, P_j)$ is defined by

$$PRWV(P_i, P_j) = \sum_{x \in P_i, y \in P_j} IRLV(x, y)$$

where, $IRLV(x, y)$ is the indirect reference level value from an object $x$ to an object $y$, and calculated by $2^{*-IRL(x, y)}$. And $IRL(x, y)$ is the indirect reference level from the object $x$ to the object $y$, which means that how many partitions are referenced for referencing from the object $x$ to the object $y$. Therefore, it can be calculated by counting the partitions through the path from the object $x$ to the object $y$ using the component referential relationships ($\nless$).

In other words, if the shortest component referential relationship path from an object $x$ to an object $y$ is $x \nless i_1 \nless i_2 \nless \ldots \nless i_n \nless y$, the $IRL(x, y)$ is calculated by subtracting 2 from the total number of partitions which contain either the object $x$, objects $i_1, i_2, \ldots, i_n$, or the object $y$. We subtract 2 to exclude the partitions which contain either the object $x$ or the object $y$. For example, if two distinct objects in different partitions reference directly each other, the indirect reference level is 0 (i.e., 2-2), and then the indirect reference level value is $2^0 = 2^0 = 1$.

The greater is the partition referential weight value between two partitions, the more frequently they reference each other. The structural clustering method stores the partitions of the partition pair which has greater partition referential weight value together as close as possible.

Let's trace this method using the example in figure 1. By the result of [Theorem 1], we may consider only some partition pairs in which the basic component set of one partition is a subset of the other. Let's prove [Lemma 1], and use it in turn to prove [Theorem 1].

[Lemma 1] If an object $x$ has an object $y$ as its component object, the basic component representation of the object $y$ is a subset of the basic component representation of the object $x$. I.e., if $x \nless y$ exists, then $P(x) \supseteq P(y)$.

Proof : By the assumption, $x \nless y$, then for all $a \in P(y), a \nearer BP$. And also by [Definition 3], $x \nless a$. Since $x \nless y$ and $y \nless a, x \nless a$. Therefore, $a \in P(x)$ and then $P(x) \supseteq P(y)$.

[Theorem 1] If there is a partition referential relationship from a partition $P_i$ to a partition $P_j$, $P_i \Rightarrow P_j$, the basic component set of the partition $P_j$, $BT_P : \{ P(x) | x \in P_j \}$ is a subset of the basic component set of the partition $P_i$, $BT_P : \{ P(x) | x \in P_i \}$. I.e., if $P_i \Rightarrow P_j$, then $BT_P \supseteq BT_P$.

Proof : By the assumption, $P_i \Rightarrow P_j$, then $\exists x, y \in P_i \wedge x \nless y$ by [Definition 5]. According to [Definition 4], if $x \in P_i$ then $P(x) = BT_P : \{ P(x) | x \in P_i \}$, and if $y \in P_j$ then $P(y) = BT_P : \{ P(y) | y \in P_j \}$. According to [Lemma 1], if $x \nless y$ then $P(x) \supseteq P(y)$. Therefore, $BT_P = P(x) \supseteq P(y) = BT_P$. And then $BT_P \supseteq BT_P$.

Therefore, it is need to be checked for calculating the partition referential weight values only when the cases the basic component set of a partition is a subset of the basic component set of another partition among all possible partition pairs. This statement means that it is not always necessary to consider all partition pairs when the referential weight values among composite objects are to be calculated.

In figure 1, we may only consider following 9 cases among all possible 30 cases (6*5 cases).

1) $(P_1 \Rightarrow P_2) : 0$
2) $(P_1 \Rightarrow P_3) : 2^0 + 2^0 = 2$
3) $(P_1 \Rightarrow P_4) : 2^0 + 2^0 + 2^0 = 4$
4) $(P_1 \Rightarrow P_5) : 2^1 + 2^1 = 1$
5) $(P_1 \Rightarrow P_6) : 2^1 + 2^1 = 1$
6) $(P_2 \Rightarrow P_4) : 2^0 + 2^0 + 2^0 = 4$
7) $(P_2 \Rightarrow P_6) : 2^0 + 2^0 + 2^0 = 4$
8) $(P_3 \Rightarrow P_5) : 2^0 = 1$
9) $(P_3 \Rightarrow P_6) : 2^0 = 1$

Let's consider case 5) as an example. The object set of partition P1 is $\{N, Q\}$ and that of partition P6 is $\{C\}$. Since the reference from object N to object C is through object T, the indirect reference level and the indirect reference level value of it is 1 and $2^{-1} = 1/2$. 

286
2 respectively. The reference from object Q to object C is similar to the former. Therefore, the partition referential weight value from the partition P1 to the partition P6 is \((1/2 + 1/2) = 1\).

We may place the partition pairs which have very great partition referential weight values together as close as possible. We call this method a partition rearrangement. The implementation of this method has two steps: 1) sort the partition pairs by their partition referential weight values, and then 2) place the partitions according to the sequence.

Let's see how this method is processed using the example in figure 1. First, the partition pairs are sorted in descending order by their partition referential weight values as follows:

6), 3), 7), 2), 4), 5), 8), 9), 1)

These partitions are placed according to the sequence as shown below.

a) place 6th pair: \((P2, P4)\)
b) place 3rd pair: \((P2, P4, P1)\)
c) place 7th pair: \((P6, P2, P4, P1)\)
d) place 2nd pair: \((P6, P2, P4, P1, P3)\)
e) place 4th pair: \((P6, P2, P4, P1, P3, P5)\)
f) place 5th, 8th, 9th pair: same to the case e)

In subchapter 5.1, we will demonstrate goodness of this method for managing composite objects.

4. Function-Propagation Clustering

After class objects clustered with previous structural clustering method, instance objects of the class objects can be clustered according to their semantics. By applying some functions supplied by users to attributes of instance objects in the same class object, we categorize the instance objects into several groups according to the values of the functions. The function-propagation clustering method is based on this idea.

The result of function-propagation clustering of a class object may be propagated to its superclass objects or its subclass objects through the composite object structure. But, the clustering propagation results in some clustering conflicts. It must be resolved.

In this chapter, we describe the overall details concerning this method.

4.1 Concepts of Function-Propagation Clustering

Function-propagation clustering is based on the functional clustering method proposed by Cheiney and Kierman to manage complex domains on a relational database system [CHEI88]. The method uses some functions applied to the domain values of relations. It categorizes all tuples of a relation into several groups according to the function values. The result of the categorization produces a kind of predicate tree. The leaf nodes of the tree contain the sets of all tuples satisfying the conditions of all predicates defined in the root node, the leaf node itself, and all internal nodes through on the path of the predicate tree.

For example, the following is a predicate tree which represents categories classified by a function "size" applied to the attributes of some relation, "length" and "height".

```
size(length) ≤ 5 5 < & ≤ 15 15 < & ≤ 50 other
size(height) ≤ 20 20 < & ≤ 30 30 < & ≤ 100 other
```

In the above figure, the length of tuples in the most left and lowest node is between 15 and 50, and the height of them is less than or equal to 20. All these tuples are to be stored together close in the secondary storage to answer efficiently all queries which access the tuples satisfying these conditions.

The function-propagation clustering is designed by adopting this functional clustering method to the environment for managing the composite objects. Let's trace the function-propagation clustering method.

The structural clustering method operates only on the class(type) objects. But each class consists of the same kinds of instance objects. Therefore figure 2 represents a structure of instance objects in the class objects of figure 1.

In figure 2, the composite structures of instance objects are shown. Then users can request a function-propagation clustering applied to the instance objects. The function-propagation clustering categorizes instance objects of a class object into several groups according to one or more functions supplied by users. The form of functions supplied by users for clustering is:

Cluster function_name(object_name) = expression applied to the attributes of the class object.

Figure 2. Composite structures of instance objects
For example, the user can define a function $f$ applied to the attributes "color" and "length" of the class object $M$ as a following:

$$\text{Cluster } f(M) = M.\text{color}[\text{red,blue}] \land M.\text{length}[^{<5,\geq5}]$$

where, items in the notation $[ ]$ are the classification criteria of the function-propagation clustering.

Then the clustering applied to a class object is propagated recursively to its subclass objects, and the instances of each subclass object are categorized several groups according to the cluster structure of its superclass object. This is a downward propagation of the function-propagation clustering.

For example, let's assume that instance objects of the class object $M$ are categorized into two groups according to the values of two attributes "color" and "length". Let's assume also that the color of instance objects $m_1$ and $m_3$ is red and the length of them is less than 5, and the properties of $m_2$ and $m_4$ are blue and greater than 5. Then the following two clusters are generated.

$$\{m_1, m_3\}, \{m_2, m_4\}$$

This cluster structure is propagated to the lower level of the class object $M$, and then the instance objects in the lower level are reclustered. The propagation is recursively continued through the path of the component reference relationships until the basic component objects are meted. The result of applying the downward propagation to the example in figure 1 is appeared in figure 3. For simplifying, we consider only the composite object $M$ and its components.

For example, let's also assume that the cluster $(m_1, m_3)$ is propagated before the cluster $(m_2, m_4)$. Then two instance objects $a_2$ and $a_3$ of class object $A$ in the child level of class object $P$ of class object $M$ construct a cluster $(a_2, a_3)$ by propagating the cluster $(m_1, m_3)$. When the cluster $(m_2, m_4)$ is propagated, it is necessary to include $a_3$ and $a_4$ into the cluster $(a_2, a_3)$ in the child level of class object $P$ of class object $M$, and then construct a new cluster $(a_2, a_3, a_4)$ by using [Propagation Rule 1].

[Propagation Rule 1] When an instance object becomes a common element of several clusters, place it into the cluster whose number of elements is greater than those of others.

For example, in figure 2, let's also assume that the cluster $(m_1, m_3)$ is propagated before the cluster $(m_2, m_4)$. Then two instance objects $a_2$ and $a_3$ of class object $A$ in the child level of class object $P$ of class object $M$ construct a cluster $(a_2, a_3)$ by propagating the cluster $(m_1, m_3)$. When the cluster $(m_2, m_4)$ is propagated, it is necessary to include $a_3$ and $a_4$ into the cluster $(a_2, a_3)$ in the child level of class object $P$ of class object $M$, and then construct a new cluster $(a_2, a_3, a_4)$ by using [Propagation Rule 1].

[Propagation Rule 2] When an instance object becomes a common element of two clusters and also the numbers of their elements are equivalent, place it into the first cluster.

For example, in figure 2, let's also assume that two clusters of instance objects of the class object $M$ by the function $f$ are $(m_1, m_3)$ and $(m_2, m_4)$. And also assume that $(m_1, m_3)$ is propagated firstly. Then an instance object $o_3$ of the class object $O$ becomes a common element of the two clusters. By [Propagation Rule 2], the instance object $o_3$ is included into cluster $(o_1, o_3)$.

Following [Lemma 2] is proved using these two propagation rules.

[Lemma 2] Propagation of the function-propagation clustering with [Propagation Rule 1] and [Propagation Rule 2] generates a nonredundant direct acyclic graph structure, in which any instance object is not occurred two or more time.

Proof: If an instance object of a class object $A$, $a$ is included into both cluster $C_1$ and cluster $C_2$ by a function at the same time, following two cases are possible:

1) When the number of elements of cluster $C_1$ and that of cluster $C_2$ are different, $a$ is included into the bigger cluster but it is excluded from the other by [Propagation Rule 1].

2) When the number of element of cluster $C_1$ and that of cluster $C_2$ are equivalent, $a$ is included into the first cluster, $C_1$ by [Propagation Rule 2].

Therefore, in any cases, the instance object $a$ is not redundantly existed in several clusters.

If we apply above two clustering propagation rules to the example in figure 3, we can get a nonredundant direct acyclic graph structure in figure 4.
4.2 Upward Propagation of Function-Propagation Clustering

Some users may want to propagate the effect of clustering to the up level of the composite object. To support it, we generalize the propagation concept with options. The effect of modifying a cluster structure is propagated to the higher level or the lower level recursively through the composite structure. The former is an upward propagation of the clustering and the latter is a downward propagation of the clustering.

It is enable us to limit the level of propagation by asserting a limit value in the cluster definition expression. The modified user requirement expression is as a follow:

Cluster function-name(object-name, options) = expression applied to the attributes of the class object-name.

where, the kinds of the options are as followings:

a) ‘r’ : propagate to the root level
b) ‘s’ : propagate to the sibling level
c) ‘+number’ : propagate up to the number level
d) ‘-number’ : propagate down to the number level

But, new cluster propagation may be conflicted to the existing cluster structures. These clustering conflicts must be resolved.

4.3 Resolution of Conflicts in Cluster Propagation

Two or more cluster propagations may be conflicted to each other. To resolve it, the following two conflict resolution rules can be used.

[Conflict Resolution Rule 1] The result of function-propagation clustering of a class object itself is prior to those of the clusterings propagated from other class objects.

For example, in figure 2, let's assume that following function g() and function h() are applied to class object R and class object S respectively before the user clusters class object M.

Clustering by g(R) : {r1,r3}, {r2,r6}, {r4,r5}
Clustering by h(S) : {s1,s2}, {s3,s4,s5}

Then, since the clustering of the instance objects of class object R may be propagated to their lower level, the cluster structure of the instance objects of class object A may be modified. And also, since the clustering of the instance objects of class object S may be propagated to their lower level, the cluster structures of the instance objects of class objects A and C also may be modified.

If the user require the function-propagation clustering of class object A by using a function f(), the cluster structure may be propagated only to the class object P, but should not be propagated to the class objects R and S which are already clustered by the functions g() and h() respectively.

[Conflict Resolution Rule 2] If an upward propagation cluster structure and a downward propagation cluster structure are conflicted(meeted) at a class object, one located close to the class object is prior to the other.

This rule can be implemented easily by decreasing the propagation level per cluster propagation. If two propagations have same propagation level, use [Conflict Resolution Rule 1].

Using these two resolution rules, we can resolve the clustering conflict problems of the subchapter 4.2. To prove this, we define the propagation length at first.

[Definition 7] Cluster propagation length from a class object X to a class object Y, PL(X,Y) is defined by

PL(X,Y) = | L(X) - L(Y) |

where, L(X) is the level of class object X in the composite structure which is a direct acyclic graph. And notation | | is the absolute function. For example, in figure 1, L(O) is 2 and L(A) is 4. Therefore, PL(O,A) is 2.

Following [Lemma 3] is established by this definition and above two conflict resolution rules.

[Lemma 3] If we cluster all composite objects by only using both [Conflict Resolution Rule 1] and [Conflict Resolution Rule 2], the cluster structure of any class object is affected only from clusters of one class object.

Proof : When a class object A is clustered by propagating the cluster structure of a class object S1 and then reclustered by propagating the cluster structure of a class object S2, following three cases are possible.
1) In case both the object S1 and the object S2 are located at the same distance from the object A (\(PL(S1,A) = PL(S2,A)\)), the cluster structure propagated from the object S1 is only accepted, but the cluster structure propagated from the object S2 is ignored according to [Conflict Resolution Rule 1].

2) In case the object S2 is located more closely to the object A than the object S1 (\(PL(S1,A) > PL(S2,A)\)), the cluster structure propagated from the object S1 is resolved and the cluster structure propagated from the object S2 is accepted according to [Conflict Resolution Rule 2].

3) In case the object S1 is located more closely to the object A than the object S2 (\(PL(S1,A) < PL(S2,A)\)), the cluster structure propagated from the object S1 is reserved, but the cluster structure propagated from the object S2 is ignored according to [Conflict Resolution Rule 2].

Accordingly, in any cases, the cluster structure of the class object A is affected by only one cluster propagation.

Therefore, if we apply above two propagation rules and two conflict resolution rules to the example in figure 4 with two functions \(g(R)\) and \(h(S)\), figure 5 is generated.

![Figure 5. Final result of the cluster propagation](image)

5. Performance Analysis

A direct acyclic graph (DAG) structure is constructed among composite objects. Two popular search techniques of DAG are the depth-first search (DFS) and the breadth-first search (BFS). Therefore, in this paper, we compare and analyse the performance of our clustering methods to access composite objects using these two search techniques. And also, to simplify the performance evaluation, we assume that the average rotation delay time is 0.01 second and the seek time between two neighbor tracks is 0.001 second.

5.1 Performance Analysis of Structural Clustering

Three methods are compared in this subchapter to analyze the performance of the structural clustering method.

Method S1 (Conventional Method): This method stores each component objects into each different partitions according to the class objects.

For the example in figure 1, there are all 11 partitions as follows:

![Partition M POS](image)

Method S2 (Kotchaba’s Method): This method stores each component objects into each partition containing several class objects. And all partitions are placed at the secondary storage in a random order.

For the example in figure 1, there are all 6 partitions as shown below.

![Partition M POS](image)

Method S3 (Structural Clustering Method): The partitioning technique of this method is equivalent to that of Method S2, but all partitions are placed at the secondary storage according to the order of their PRWVs.

For the example in figure 1, there are all 6 partitions as follows:

![Partition M POS](image)

To measure the performance of these three methods, we generated four random DAGs of composite objects and evaluated the average access times of composite objects of each DAG in each methods. The numbers of nodes of each DAG were 50, 100, 150, and 200. The average fan-outs and fan-ins of each node were 3...
and 5 respectively. The result of experiment is represented in figure 6.

![Figure 6. Comparison of structural clustering methods](image)

As you can see the result in figure 6, both Method S2 and Method S3 are superior to Method S1 in which no structural clustering is applied. Method S3 is superior to Method S2 at the average access time. This is because, using the two clustering methods, the related components to be retrieved together are located in either an equivalent page or the neighbor pages, and then fewer page I/Os or less seek times are required. But, we can find a few cases in which Method S2 is superior to Method S3. This indicates that our clustering method (Method S3) is not always superior to Method S2. This is because that there are very complicated relationships among composite objects in the DAG structure but our method (Method S3) is considered only one relationship among them. But, in general, Method S3 is the best among three.

5.2 Performance Analysis of Function-Propagation Clustering

Three methods are compared also in this subchapter to analyze the performance of the function-propagation clustering method.

Method F1 (Conventional Method) : This method stores all component objects into each different partitions such as the Method S1, but it does not use any function-propagation clustering.

For the example in figure 5, the configuration of partitions is equivalent to the case of Method S1 as followings:

<table>
<thead>
<tr>
<th>Partition M</th>
<th>Partition P</th>
<th>Partition O</th>
<th>Partition R</th>
</tr>
</thead>
<tbody>
<tr>
<td>{m1,m3}</td>
<td>{m2,m4}</td>
<td>{o1,o3}</td>
<td>{r1,r3}</td>
</tr>
<tr>
<td>{m3,m4}</td>
<td>{m1,m2}</td>
<td>{o2,o4}</td>
<td>{r2,r4}</td>
</tr>
<tr>
<td>{s1,s2}</td>
<td>{s3,s4}</td>
<td>{a1,a2}</td>
<td>{c1,c2}</td>
</tr>
<tr>
<td>{s3,s4}</td>
<td>{s1,s2}</td>
<td>{a3,a4}</td>
<td>{c3,c4}</td>
</tr>
<tr>
<td>{s1,s2}</td>
<td>{s3,s4}</td>
<td>{a5,a6}</td>
<td></td>
</tr>
<tr>
<td>{s3,s4}</td>
<td>{s1,s2}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Method F2 (Function-Propagation Clustering Method) : This method stores all component objects after applying the function-propagation clustering to them.

For the example in figure 5, there are all 11 partitions and all instance objects in each partitions are clustered as followings:

<table>
<thead>
<tr>
<th>Partition M</th>
<th>Partition P</th>
<th>Partition O</th>
<th>Partition R</th>
</tr>
</thead>
<tbody>
<tr>
<td>{m1,m3}</td>
<td>{p1,p3}</td>
<td>{o1,o3}</td>
<td>{r1,r4}</td>
</tr>
<tr>
<td>{m2,m4}</td>
<td>{p2,p4}</td>
<td>{o2,o4}</td>
<td>{r2,r6}</td>
</tr>
<tr>
<td>{s1,s4}</td>
<td>{s2,s5}</td>
<td>{a1,a6}</td>
<td>{c1,c3}</td>
</tr>
<tr>
<td>{s3}</td>
<td></td>
<td>{a5,a7}</td>
<td>{c2,c4}</td>
</tr>
</tbody>
</table>

Method F3 (Combination Method) : This method stores all component objects after applying both the structural clustering and the function-propagation clustering.

For the example in figure 5, there are all 6 partitions and all instance objects in each partition are clustered as followings:

<table>
<thead>
<tr>
<th>Partition C</th>
<th>Partition MPOS</th>
<th>Partition RA</th>
</tr>
</thead>
<tbody>
<tr>
<td>{c1,c3}</td>
<td>{r1,r4,a1,a6}</td>
<td>{r3,r5,a7}</td>
</tr>
<tr>
<td>{c2,c4}</td>
<td>{r2,r6,a2,a3}</td>
<td></td>
</tr>
</tbody>
</table>

We simulated these three methods and evaluated average access time of composite objects in each method under previous experimental parameters. The result of the performance evaluation of these three methods is represented in figure 7.

![Figure 7. Comparison of function-propagation clustering methods](image)

As shown in figure 7, both Method F2 and Method F3 are superior to Method F1 in which no clustering is applied. Method F3 is superior to Method F2, but the improvement is not satisfactory. However, if almost all patterns of queries are similar to the cluster conditions of the function-propagation clustering, the
performance of Method F3 will be clearly better than that of Method F2.

6. Conclusion and Future Works

We suggested two clustering methods for managing composite objects efficiently: the structural clustering method and the function-propagation clustering method. The structural clustering method is an extension of the Ketabchi's component aggregation concepts, and the function-propagation clustering method is made from the Cheiney's functional clustering method.

Comparing to the several clustering methods, we knew that our two clustering methods are efficient for managing composite objects, and especially the combination of the two clustering methods is very efficient for managing them in the object-oriented systems.

To use the results of this study more well, following fields must be studied thoroughly.

First, to construct the IRL function of the structural clustering correctly, we must reflect the effect of the number of instance objects in each class object.

Second, we must consider other composite relationships to construct more efficient partition rearrangement method. And also it is possible to use the composite relationships among three or more partitions for extending the partition rearrangement technique.

Third, we must consider the continuous evolution of the number of composite objects at our two clustering methods.

References


