12-31-1993

Clustering Objects for OODBs in a Multiple Relationship Environment

Steven Tu  
Massachusetts Institute of Technology

Daniel Buehrer  
National Chung Cheng University

Follow this and additional works at: http://aisel.aisnet.org/pacis1993

Recommended Citation
http://aisel.aisnet.org/pacis1993/28

This material is brought to you by the Pacific Asia Conference on Information Systems (PACIS) at AIS Electronic Library (AISeL). It has been accepted for inclusion in PACIS 1993 Proceedings by an authorized administrator of AIS Electronic Library (AISeL). For more information, please contact elibrary@aisnet.org.
Clustering Objects For OODBs in a Multiple Relationship Environment

Steven Yi-Cheng Tu
Composite Information Systems Laboratory
E 33-318 Sloan School of Management
Massachusetts Institute of Technology
Cambridge, MA 02139 U.S.A.

David J. Buehrer
Institute of Computer Science and Information Engineering
National Cheng Cheng University, Chia-Yi, Taiwan, R.O.C.

Abstract

When data objects in databases are involved in more than one kind of relationship, we say that these objects are in a multiple relationship environment. Object-Oriented databases are an example of such an environment. In this paper, we explore how to cluster objects in secondary storage. We develop a distance metric for objects to measure the degree of inter-relationship, and apply a greedy algorithm to order the objects. After this sequence is constructed, we can easily find the object sequence on the secondary storage and achieve the purpose of clustering. We also discuss in this paper the features of our method that make it different from the previous proposed methods.

1. Introduction

1.1 Overview of Clustering in OODBs

The Object-Oriented (OO) paradigm has been recognized as a useful approach in knowledge representation, programming languages, and software engineering. Besides, the OO concept has also been applied to the database systems recently because of its expressibility and semantic richness. Object-Oriented databases (OODBs) offer the next-generation database applications a good solution to the major shortcomings of traditional relational database systems. These are several characteristics of OODBs which we consider make it especially suitable for the newly-emerging applications, such as OIS, CASE, and CAD/CAM systems. These characteristics include 1) the support of complex objects that enable users to define new complex data types instead of only system-defined flattened primitive data types, 2) the inclusion of abstract data types which make it easier to describe the dynamic behavior of the data, thus increasing the level of abstraction, 3) the use of inheritance that facilitates the database designers to reuse the attributes/code definitions, and 4) the support of object identity that makes it more natural to model the real-world entities.

Many OO languages such as Smalltalk and CLOS have been extended with the functions of persistence and shareability in order to make them become database programming languages. These OO languages have both the advantages of object-orientation and database systems. However, in order for these Object-Oriented database systems to be more practical and useful, some other database features must be added [9][23][24]. For example, a declarative query language [18], concurrency management [14], authorization [12], and storage control [13] must also be reconsidered in OODBs. Especially, the functionality of managing large amount of data safely and efficiently is crucial for OODBs design. In this paper, we concentrate on discussing and proposing an approach for clustering in OODBs.

Clustering is an important issue in physical database design [15]. It is concerned with how to move the related data onto secondary storage as closely as possible in order to expedite the efficiency of data retrieval. Therefore, clustering can improve the performance of the database systems, which is the major problem of OODBs. In traditional relational database systems, each relation is independent. The relations are joined dynamically according to the inter-relationship of relations when a user's query is processed. Consequently, it is sufficient to use either the n-ary storage model (NSM) or the decomposition storage model (DSSM) [3][14] for the relational database clustering. But in OODBs, clustering is complicated by the multiple relationships among the objects. That is, an object can be both an instance of a class and a component of a complex object. Besides these aggregation and generalization relationships, other structural relationships such as version and configuration [1] can also exist. In this situation, the objects are in a multiple relationship environment. This environment affects the clustering methods that can be used because we have multi-dimensional goals while storing the related objects onto the single-dimensional storage. In order to solve this problem, we propose a method in this paper for compromising on the possible relationships among objects and then ordering the objects into a clustering sequence. After the sequence is constructed, we can then easily map the sequence into the secondary storage and preserve the multiple relationships of the objects, thus achieving the purpose of clustering.

1.2 Previous Related Work

Most previous papers on OODB clustering have been focusing on complex objects (nested objects). In [2] Banerjee and Kini studied the possible universal methods of a DAG (Directed Acyclic Graph), which is in fact the complex object hierarchy. In [3] Dewitt discussed the representation of large complex objects and the algorithms to dynamically insert and delete objects in the large complex objects with an emphasis on optimizing performance. Another complex object storage model which is not based on the fact object reference relationships but rather on the object identifiers was presented in [23]. But the above research only restricted their discussions to a single relationship between objects and did not cover the thorough semantics of the OOP paradigm. [1] was the first paper to take into account the multiple relationships (Inheritance, Version, Configuration) in OODBs, and a smart clustering algorithm was also given that...
could exploit these structural relationships for clustering objects. However, only one kind of relationship could be input into the smart algorithm instead of all possible multiple relationships. The actual pioneering paper dealing with the clustering techniques in a multiple-relationship environment is [3]. In [5], related objects can be connected into a multi-graph by assigning a weight according to each relationship between two objects. Then, a level clustering algorithm is used to construct a maximal spanning tree from this weighted multi-graph. The clustering problem and our approach in this paper must resemble [5]. However, our approach is different from [5] in that we actually develop a metric to measure the distance between objects, which is a concrete calculation of the weight w in [5]. Besides, the distance metric in our approach also takes into account the cardinality (size) of each relationship set (explained later), which was not considered in [5].

The remaining sections of this paper are organized as follows: In Section 2 we give an OODB example model in order to clarify our problem and also pave the way for future discussions. Section 3 presents our proposed approach, including the distance metric and sequencing algorithm. In Section 4, we discuss the features of our approach and also give some conclusions.

2. The Example Model and Clustering Options
In order to better explain the clustering options in a multiple relationship environment and use this example as an illustration when we introduce our approach, we borrow a sample model from [8]. We choose this example for three reasons. One reason is that most of the important concepts of OO are incorporated into ORION [19], which make it a better each OODB. Another reason is that most of the major issues of ORION, including the query model [20], schema evolution [15], and transaction control [16] are already discussed in the literature. The third reason is that ORION system is not only a proposed model but also implemented and operational [20].

The following graph is a schema example of a multi-table database of an OODB. Each square box is a class with attributes for that class defined inside the box. The dashed line represents the inheritance (is-a) relationship and the solid line represents the aggregation (part-of) relationship. The part-of relationship is the inter-object relationship or the above-mentioned complex object relationship, and the is-a relationship says that each instance of a subclass is also an instance of its superspecies. Another relationship which is not explicit in this example is the instance-of relationship. All of the instances belonging to the same class are considered to possess this relationship. In this paper, if only one kind of relationship was considered, we call that clustering in a single relationship environment, and if all relationships are considered, then clustering is in a multiple relationship environment.

Several clustering options for the above sample database may be adopted. Option 1 is to cluster all objects which belong to the same class. This involves storing objects according to the instance-of relationship. For example, all objects in Vehicle would be stored together. Option 2 is to cluster all objects belonging to the same class hierarchy. This involves storing objects according to the is-a relationship. For example, all object in Vehicle, Automobile, and Track would be stored together. Option 3 is to cluster objects that other object inter-reference. This involves clustering objects according to the the part-of relationship. For example, the objects in Vehicle, Company, and Employee would be stored together. It should be noted that the above clustering options are just some of the possible relationships in OODBs. Other more complicated relationships from applications' semantics might also exist. For example, Figure 2 represents the version relationship for the sample database.

![Figure 2: A version network](image)

Each clustering option has major effects on the retrieval performance for the application's access pattern. The access pattern is defined as the traversal sequence for the classes involved in the user's query. We list several possible user's queries to the sample database. A class variable associated with a star (*), like Vehicle, means to retrieve from this class and all the subclasses rooted at that class.

Query 1: (Vehicle select V (V color = "blue")
Query 2: (Vehicle select V (V drivetrain transmission = "fuel" and V manufacturer name = "Ford"))
Query 3: (Vehicle select V (V weight > 1000 and color = "red"))
Query 4: (Vehicle select V (V color = "blue") and (V manufacturer name = "Ford"))

For each query above, there is a corresponding clustering option that makes the retrieval performance of that query some efficient. For example, the clustering option 1 favors query 1. Similarly, clustering option 2 favors query 2 and clustering option 3 favors query 3. However, no clustering option that we mentioned above favors query 4 because it involves more kinds of relationships than other queries. In summary, each clustering option might be optimal for some kinds of access patterns (queries) but may sacrifice performance on other access patterns. Furthermore, deferring the clustering to one kind of relationship will incur very poor access performance for other queries. For instance, clustering objects according to part-of relationship in the same disk block will cause objects of any single class to be spread out over a larger number of pages than if the class is the only class to be stored in one block. However, the multiple relationship property is an inherent constraint for both Object-Oriented systems and next-generation applications. Therefore, some way to compromise all of these relationships while dealing with clustering must be developed in order to achieve a better average performance for users' ad hoc queries. This is the motivation for our method to solve this problem.

3. The Distance Metric and Clustering Algorithm

![Figure 1: A sample database](image)
3.1 Outline of Our Approach

As mentioned before, we are going to use distance to measure the inter-relationship of objects. Each object, according to the kind of relationships in which it may participate, can be represented by an n-tuple of (category 1, category 2, ..., category n ). If there is no possible relationship involved in the system. For example, an object of ( C1, COI, VI) can be thought of as belonging to class C1, complex object COI, and version number VI. One problem is that most of the values of each category in the n-tuple are qualitative and discrete data. This means that the distance between each two objects, some quantifying method must be used in order to convert the n-tuple into numeric data, from which a distance can be measured. There are two basic constraints which we consider must be satisfied if some quantifying method is to be used. The first constraint is that any quantifying configuration which faithfully represents the patterns existing among objects must guarantee that more related objects should be represented by points that are closer. The second constraint is that if two objects are not-comparedly inter-related, then there should not be any less or greater distance comparison after the transformation. These two constraints can be represented as follows where $\Theta$ denotes the semantic relationship between objects and $\alpha$ denotes the closer than operator.

Constraint 1 : $\Theta(01,02) \land \Theta(01,03) \Rightarrow \text{Distance}(01,02) < \text{Distance}(01,03)$

Constraint 2 : $\neg ( ( \Theta(01,02) \land \Theta(01,03) ) \lor ( \Theta(01,02) \land \Theta(01,03)) ) \Rightarrow (\text{Distance}(01,02) < \text{Distance}(01,03) )$ or (Distance(01,02) < Distance(01,03))

Our original idea was to quantify each object from a qualitative n-tuple to a numeric n-dimensional coordinate system and apply a multivariate analysis method such as the principle component analysis [7] to project these n-dimensional points into the principle components, and to maximize the preservation of relationships from the original n-dimensional space. We did not adopt this approach because when we quantize each object into an n-dimensional numeric coordinate, we lose some semantics. This will not satisfy the second constraint. For example, if category 1 represents the class to which one object belong and we quantize class Vehicle as 1, Company as 2 and VehicleDriver as 3, then this makes Vehicle more related to VehicleDriver than to Company (3-1 > 3-2). However, this quantified transformation of the qualitative data is wrong. Therefore, we develop a metric to directly measure the distance between each two objects instead of transforming each object into an n-dimensional point. Our metric satisfies both the first and second constraints, thus does not lose any semantics of object inter-relationships.

3.2 Distance Metric

A relationship set is defined to be the set of objects that are constrained in the relationship and the size of the relationship set is defined to be the number of objects in this set. For example, one relationship set is the class Vehicle, and its size is the number of instances in Vehicle. Another relationship set is the complex object instance involved in Vehicle, Company, and Employee, and its size is the number of objects that comprise this complex object instance. In this paper, each relationship set will be denoted Rm and its size will be denoted |Rm|.

Distance$(o, o_j)$ =

$$\sum_{m=1}^{k} \frac{1}{b} \left[ \delta_{i,j,Rm}(1/2) \cdot |Rm| + \beta_{i,j,Rm}(1/2) \cdot L \right] \cdot x$$

where

- $m$ : number of relationship sets
- $k$ : number of objects in the system
- $s$ : object sizes
- $L$ : total size of all objects ($L = k \cdot s$)
- $b$ : disk block size

The above formula is rather generalized since it encompasses every possible relationship in the system and is calculated by compromising the effects of all relationships. In this formula, we have assumed that the size for each object in the system is the same, and is equal to $s$. Pm is the probability that the system is clustered by relationship Rm. This probability comes from the probabilistic distribution of application access patterns and can be specified by system designers. If some access pattern occurs more often or is more important, then its corresponding probability will also be larger. However, we assume that each probability Pm is the same and is equal to 1/n in the discussion below. We now explain the meaning of this distance metric. It is the expected value of the object distance by considering all semantic relationships. This expected distance is formed by summing the product of the probability Pm and the average distance (see Figure 3) of objects clustered by each possible relationship. The $R_m$ is the m-th relationship set or the property category in if objects are represented by an n-dimensional coordinate. $\delta_{i,j,Rm}$ and $\beta_{i,j,Rm}$ are mutually inverse characteristic functions that have values either 1 or 0. They can be stated as follows:

$$\delta_{i,j,Rm} = 1$$ if object $i$ and object $j$ are in relationship set $m$

$$\beta_{i,j,Rm} = 0$$ if object $i$ and object $j$ are not in relationship set $m$

For the simple case that only instance-of and part-of relationships exist in the system, Suppose that the instance-of relationship is indexed as $R_1$ and part-of relationship is indexed as $R_2$. We show how the distance of each two objects of the example in Section 2 can be calculated. Because we assume that $P_1$ and $P_2$ are the same, they will be both equal to 1/2.

Case 1 : $1/2 \times |L| \times 1/2 \times 1/2 \times L$ if 0 and 1 are in the same class $C_r$ but not in the same complex object

Case 2 : $1/2 \times |L| \times 1/2 \times 1/2 \times |C_o|$ if 0 and 1 are not in the same class but are in the same complex object $C_o$

Case 3 : $1/2 \times |L| \times 1/2 \times 1/2 \times 1/2 \times |C_o|$ if 0 and 1 are in the same class $C_r$ and are also in the same complex object $C_o$

Case 4 : $1/2 \times |L| \times 1/2 \times 1/2 \times |L| \times 1/2 \times L$ if 0 and 1 are not in the same class and are not in the same complex object

For example, the distance between two objects in class Vehicle will be $1/2(1/2 \times |Vehicle| \times 1/2 \times L)$ if they are not
cyclically referenced. However, if $o_i$ and $o_j$ are further recursively referenced (1-1), then the distance will be $1/2\cdot1/2$ (CQ) where CQp is the complex instance that contains $o_i$ and $o_j$. It can be shown that the distance for the above case has the property that distance (case 3) $< \text{distance(case 1)} < \text{distance(case 2)} < \text{distance(case 4)}$. This satisfies our constraint 1 because more related objects have less distance. However, the distance for case 1 and case 2 is non-comparable. It depends on the size of the relationship set (X3 and X2). This also satisfies our constraint 2 because the closeness relationship set is not comparable. If $o_i$ and $o_j$ are in the same class and $o_i$ and $o_j$ are in the same complex object, then the absolute semantic closeness can be stated for $o_i$ to $o_j$. The two constraints will be satisfied because the expression $|R_i| < 1$ holds for every relationship set in the system. We now explain how the relationship size is included in the formula. This can be demonstrated in Figure 3. If two objects are in the same class and we also cluster by the instance of relationship, then the distances will be the average of the class size. Figure 3 explains how the distance between $o_i$ and $o_j$ is obtained for Case 1.

![Figure 3 Clustering options for a two-relationship simple case](image)

A similarity matrix [24] can then be constructed. This is a $k \times k$ matrix where each entry of the matrix is the distance between any two objects ($M(s) = \text{distance}(o_i, o_j)$). This matrix is symmetric with all diagonal elements equal to 0. From this matrix, we can then find a spanning tree which includes all of these $k$ objects by applying the sequencing algorithm in the next section.

3.3 Sequencing Algorithm

Since we must eventually put all of the objects onto the linear secondary storage, a better way is to sort the objects of the system into a one-dimensional data structure. Then the mapping of the one-dimensional data structure to the secondary storage will be straightforward. The previous research in [5] constructed objects into a maximally weighted spanning tree. However, because a tree is a two-dimensional data structure, some traversal methods such as DFS or BFS must be applied in order to linearize the spanning tree. We consider this to be disadvantageous because when we linearize the spanning tree, some distance relationships, which are also semantic relationships of objects, may be compromised. Therefore, our sequencing algorithm outputs the result as a spanning path, by which all of the multiple relationships can be still preserved to some degree. After the distance matrix is constructed by assigning the distances between each two objects, the following sequencing algorithm [11] can be used to order the objects into a path sequence. The input of this algorithm is the distance matrix which can represent the weighted average of each relationship between each two objects. We also assume that there are totally $k$ objects in the system. Each of these $k$ objects can be a member of more than one kind of relationship set. This algorithm is based on the greedy strategy because it works by selecting one object at a time into the current spanning path. This can be done by selecting an unprocessed object which results in the minimum distance to the total distance of the current spanning path.

![Figure 4 Spanning Path Algorithm](image)

**[Totally Shorter Spanning Path Algorithm]**

1. **Input:** Similarity matrix $M(s)$.
2. **Output:** A spanning path that includes $[o_1, o_2, \ldots, o_k]$.
3. **Step 1:** Let $i = 1, W = \{o_1, o_2, \ldots, o_k\}$, and current spanning path $P = \{\}$.
4. **Randomly select an object $o'$ from the unprocessed objects $W$.**
5. **Delete $o'$ from $W$.**
6. **Step 2:** Choose an object $o'$ from the remaining $k$ objects in $W$ such that $M(R_i, o_i) \cdot M(o_i, o_i + 1) \cdot M(R_i, R_i + 1)$ is minimized.
7. **Insert $o'$ into the current spanning path to be $P = [R_i, R_i + 1, R_i + 1, o_i, R_i + 1]$.**
8. **Delete $o'$ from $W$.**
9. **Step 3:** If $i < k$ then $i = i + 1$, go to Step 2.
10. **Step 4:** Output $P$ as the clustering sequence.

The problems of finding the optimal clustering schemes for generalized access patterns and the finding of a **Shortest Spanning Path (STSP)** are both NP-Complete [11][17]. Our **Totally Shorter Spanning Path (TSSP)** algorithm was inspired by the **Shorter Spanning Path (SSP)** algorithm proposed in [11]. The main difference between TSSP and SSP is that in Step 3 SSP randomly chooses an object from $W$ and inserts it into the current spanning path at a particular position which adds the minimum extra distance but TSSP iterates over all of the unprocessed objects in $W$ to find the minimum extra distance. The total distance of the resulting spanning path for TSSP is less than that of SSP because in TSSP the checking of all of the unprocessed objects in $W$ also covers the checking of the randomly selected object from $W$ in SSP. In order to clarify the difference between these three spanning path constructing algorithms, we compare STSP, TSSP, and SSP with respect to time complexity and optimality in the following:

| Optimality | STSP $>$ TSSP $>$ SSP |
| Execution Time | $O(nk^2)$ $>$ $O(k(k - 1))$ $>$ STSP (exponential) |
Because the secondary storage access unit is a disk segment or a block, the spanning path must be cut into path segments according to the ratio of the object size to the block size (63). Notice that our clustering sequence is not interrupted by segmenting the spanning path. The change only occurs at the segmentation boundary. This can still be compensated for by placing the adjacent path segments into the adjacent neighboring blocks, which can save disk access time.

In the next section, we discuss the common features of clustering in database systems and compare our approach with the previous proposed methods.

4. Discussion of Our Approach and Conclusions

4.1 Discussion

In this section, we discuss several features of our approach, which are also the criteria for clustering. We only list those points that we consider different from previous proposed approaches.

1. Access performance: Clustering speeds the access performance of related data in the database. The related data are correlated by multiple kinds of relationships. Therefore, the access performance is the expected average access cost of all related access patterns, each of which retrieves data according to some relationship. Because there are a kind of relationships in the system, a kind of access patterns (queries) denoted as $Q_{R_1}Q_{R_2}...Q_{R_n}$ may be issued to the databases. We also assume that each clustering scheme based on relationship set $S$ favors access pattern $Q_{R_m}$. The retrieval cost of objects for $Q_{R_m}$ will be proportional to the distance of any two objects belonging to relationship set $S$. If all of the access patterns are considered, then the average retrieval cost for $Q_{R_1}Q_{R_2}...Q_{R_n}$ and $Q_{R_m}$, which equals $P_{f} \cdot \text{Cost}(Q_{R_1}) + P_{f} \cdot \text{Cost}(Q_{R_2}) + ... + P_{f} \cdot \text{Cost}(Q_{R_n})$, will be proportional to our distance metric. If the access probability of $Q_{R_m}$ is $P_{f}$. This means that our distance metric is also based on the expected average access cost for possible access patterns $Q_{R_1}Q_{R_2}...Q_{R_n}$ and $Q_{R_m}$. Our approach gains in access performance in that we are constructing a totally shorter spanning path that sums up less total access cost. Therefore, our approach has a better chance to induce better average access performance. We want to emphasize that the optimal access performance is very hard to achieve because it's an intractable problem. The advantage of our approach comes from the merits of the shorter spanning path (SSP), which are shown in [11]. Furthermore, our sequencing algorithm results in a TSSP which is better than SSP in [11].

2. Automatic and adaptive: Many DBMS adopt the scheme of receiving hints from users, which are frequent access patterns, and support a default strategy if hints are not specified by users. Two drawbacks might exist for this scheme. One is that users have to deal with the multiple relationships of objects, which are intrinsically multi-dimensional and which could even conflict with each other. Users probably are able to specify one kind of relationship but are unable to combine all of the relationships and then give hints to the DBMS. The other problem is that sometimes users will need to specify a subgraph of the need schema graph of a class for clustering (26). For example, it may be useful for users to give hints to store physical components of a vehicle object in the same block, but not the Company objects for the manufacturer attributes of the vehicle objects. This problem comes from the fact that users have little knowledge of the extended information, such as the class size, the actual object references relationships and object sharing relationships, of the databases. Therefore, a better approach is for the system to automatically cluster objects, because the system knows more information for clustering than the users. Our approach is automatic because the calculation of the distance metrics and sequencing process are all done by the system. However, our approach is also adaptive. This is from our sequencing algorithm Step 2 that when there are several chosen objects that all satisfy the condition ($MR_{R_1}\cdot Q_{R_2}...Q_{R_n}$), then a grouping was made.

3. Reordering: Reordering in databases is used to divide the database into homogenous groups and store each homogeneous group in a multi-field system or distributed environment in order to exploit the parallelism of distributed data (45)(6). Our sequencing algorithm outputs the result as a path sequence. This sequence can then be cut to form homogenous groups, which we call path segments above. Each path segment can be allocated to different environments or to each block in a multi-field system and a parallel search technique may be used to fetch the access speed for the users' queries (6).

4.2 Conclusions

A method of measuring distances between objects and a sequencing algorithm are proposed in this paper. Our distance metric covers the thorough semantic relationships of objects in the OODBS. It is important for Object-Oriented database clustering because the OODBS applications impose more complicated access patterns than traditional relational database applications. Any clustering schema for OODBS which is offered to versatile applications should not favor some special access pattern but sacrifice others. A better average performance for each kind of access pattern might be achieved in order to generalize the uses of the Object-Oriented database system. Our method is based on the simplicity of calculating distance but without losing flexibility to consider multiple kinds of object relationships. It is further enhanced by our sequencing algorithm, which finds a direct mapping between the object sequence and secondary storage. Because of the importance of improving the efficiency of OODBS for distributed systems, we feel that clustering in multiple-relationship environments should remain a major research issue for some time to come.

References


[16]. Kim Kyung-Chang, Won Kim, "Cyclic Query Processing In Object-Oriented Databases", *IEEE Intl. Conf. on Data Eng.*, 1989.


[26]. Won Kim, "Object-Oriented Databases: Definitions and Research Directions", *IEEE Trans. on Knowledge and Data Eng.*, Vol 2(3), Sep., 1990, pp.327-341.