A COMPARISON OF TWO REPRESENTATIONS FOR COMPLEX OBJECTS

by

Anant Jhingran and Michael Stonebraker

Memorandum No. UCB/ERL M90/32

27 April 1990

COVER PAGE

A COMPARISON OF TWO REPRESENTATIONS FOR COMPLEX OBJECTS

by

Anant Jhingran and Michael Stonebraker

Memorandum No. UCB/ERL M90/32

27 April 1990

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

TITLE PAGE

A COMPARISON OF TWO REPRESENTATIONS FOR COMPLEX OBJECTS

by

Anant Jhingran and Michael Stonebraker

Memorandum No. UCB/ERL M90/32

27 April 1990

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

A Comparison of Two Representations for Complex Objects

Anant Jhingran and Michael Stonebraker
Computer Science Division
University of California at Berkeley

Abstract

In this paper two complex object representation techniques are compared from a performance perspective. In *procedural* representation, the set of sub-objects of an object is represented intensionally using database queries. In contrast, *OID representation* uses an extensional approach. We identify four dimensions that illuminate the differences between these approaches: cost of accessing the sub-objects, clustering, caching, and query modification. We study these dimensions using mostly analytical techniques, though some simulation results are also presented. We demonstrate that an intensionalIn particular, it is shown that clustering the objects and sub-objects together is an effective tool, provided a sub-object is shared by only a few objects. Both caching and query modification are shown to benefit procedural representation more than OID representation.

1. Introduction

With emerging new applications (e.g., CAD [BATO85, LORI85], Office Information Systems and Logic Programming [ZANI85]), database systems are being asked to provide efficient support for complex objects. This complexity may arise from IS-PART-OF relationships (as in VLSI cells where "rectangles" and "paths" are parts of "cells" [LORI85]). In contrast, set-valued aggregation as in [SMIT77], where attributes of objects are themselves other objects, provides another mechanism for supporting complex entities in a database. As an example of the latter dimension of complexity, consider a complex object "department" which has the following schema in a syntax similar to EXTRA [CARE88]:

name = char[16], manager = char[16], location = char[16], employees = {**ref** employee}

The last attribute refers to the employees working in the corresponding department, and is a set-valued aggregation of the entities of type "employee".

There have been numerous proposals for representing complex objects [COPE85, BANE87, KIM87]. However, these studies have concentrated on one or two representations, with few looking at the trade-offs between many alternatives. For example, the emphasis by a group at MCC [COPE85, VALD86] is on a "decomposed storage model" of complex objects. The accompanying performance data is used to justify their choice. In a series of papers (e.g. [BANE85, KIM87]), the ORION group at MCC has dis-

This research was sponsored by the National Aeronautics and Space Administration under grant NAG 2-530, and by the Army Research Office under contract DAAL03-87-G-0083.

cussed performance implications of representing complex (especially VLSI) objects in an object-oriented environment. Furthermore, almost all semantic models have mechanisms to represent complex objects, but the work there concentrates more on data modeling than on performance issues [YAO85]. Lastly, in [SHEK89], a group at Wisconsin has studied some of the performance implications of representing complex objects using EXTRA, the data model of EXODUS [CARE88].

In [JHIN90a] our approach has been more general. We have established a framework for modeling and then comparing the various alternatives for representing the relationships between objects and sub-objects. Broadly speaking, there are three possibilities here, and they are termed **primary** representations. To illustrate the differences, consider complex objects of type "department" discussed above. Table 1 illustrates how the three primary representations differ in representing the members of the "Toy" department.

Representation	Instance		
Procedural	retrieve (employee.all) where employee.dept = "Toy"		
OID	2314, 4562, 9874, ...		
Value-Based	John	23	M
	Mary	42	F
	Doug	24	M
	...	...	...

Table 1

In **procedural** representation, the set of sub-objects associated with an object is identified "intensionally," using a database procedure. This procedure is generally a sequence of retrieve statements in the query language of the database system, which when executed return the values of the sub-objects.[1] Assuming that entities of type employees have an attribute "dept" containing the name of the department the employee works in, a possible procedure for the members of "toy" is shown in Table 1. In this representation, insertions (deletions) of sub-objects require no modification to the objects, since the procedures can re-compute the set of sub-objects on demand. POSTGRES [STON86] is an example of a database system that allows procedure as a data type.

[1]The value of an object is simply the concatenation of the values of each of its attribute.

In contrast, both **Object Identifier** and **Valued Based** representations use an extensional approach to identifying the sub-objects. In the former, a unique location independent identifier [KHOS86] for each sub-object is stored with the object. Since in general an object may have more than one sub-object, this results in a list of identifiers (OID-list for short) being stored with the object. In Table 1, the list represents the OIDs of the employees working in the Toy department. Most object-oriented database systems (e.g., ORION [BANE87], GemStone [COPE84]) use this approach to modeling complex objects.

Finally, in the value-based model, the sub-objects have no identities of their own; their values are stored with the objects that refer to them. Assuming that the relevant attributes of an employee are name, age and sex, the value of the members of "Toy" is shown in Table 1. Most non-normalized databases (e.g., AIM-II [DADA86]) model complex objects using value-based representation. This model is presented here only for the sake of completeness; it will not be discussed further in this paper.

The three representations are not totally equivalent -- they have some semantic differences. In both the extensional models, any insertion of sub-objects requires modifications to the objects that will reference these new sub-objects. Similarly, deleting a sub-object results in updates to the OID-list or values stored with all the objects that referenced it. Also, in order for two objects to share a sub-object, its identity or value has to be replicated in both places. This is in contrast to procedural representation where sharing will be implicitly achieved through the procedures stored with the two objects.

Very often, queries on complex objects have to be processed by traversing each complex object top-down. This involves efficiently determining the "values" of the component sub-objects, and is termed **materialization** (and we use the terms "materialized result" and "value" of an OID-list or a procedure interchangeably). In procedural representation, materialization requires the execution of the corresponding procedure. In OID representation, we have to dereference each OID in the corresponding OID-list to get to the location (and hence the value) of the sub-objects. Both these operations may prove to be quite expensive. There are three ways that these materialization costs can be minimized.

(1) Sub-objects can be clustered with the objects that reference them. This involves assigning objects and sub-objects to buckets such that most objects have their sub-objects in the same bucket as them.

(2) The materialized result can be computed once and stored separately on the disk. This "materialize and cache" strategy is shown to be effective for both procedural and OID representations in [JHIN88, JHIN90a].

(3) When complex objects are modeled using special parameterized procedures ([ROWE87, JHIN88]), it is often possible to rewrite the user-submitted queries such that top-down processing is no longer a pre-requisite, and hence materialization costs do not enter the picture.

The background for this study has been laid in [JHIN88] where we analyzed the performance implications of procedural representation, and in [JHIN90a] where we did a similar analysis for OID representation. Each of these studies explored the various possibilities on one representation only; no attempt was made to contrast the two primary representations. In this paper, however, we compare procedural and OID representations on four dimensions which elucidate the differences between them.

The rest of the paper is organized as follows. Section 2 discusses the first axis for comparison, namely the I/O cost of materialization. The next three sections discuss the performance of the three strategies (namely, clustering, caching and query modification) used to reduce the materialization costs. Finally the paper ends with some conclusions in Section 6.

2. I/O Cost of Materialization

We first look at the materialization costs in OID representation. When an OID-list contains s OID's, this involves dereferencing s pointers to get to the physical location of the objects. If the physical location of an object is computable from its OID (either directly, or through a hashing function), then materialization involves at most s data page accesses. However, it is often desired that objects can be moved independent of their OID's [KHOS86]. This in turn implies that OID's are location independent surrogates, and hence an auxiliary structure is required to dereference them. We next calculate the materialization costs in OID representation, assuming that the auxiliary structure is an OID-index in the form of a B+-tree. Given an OID, this index will give the physical location of the corresponding object.

Consider the following parameters of the database:

H: Height of the OID index (levels numbered [0..H]).
B: Branching factor of the index
M: The number of sub-objects in each data page

N: Number of sub-objects in the database
s: Number of entries in the OID-list

In general, the s OID's in an OID-list are drawn randomly from the set of N possibilities. To see why this is the case, consider the following schema:

PERSON (name, profession, city)
PROFESSION (name, members)
CITY (name, RESIDENTS)

The attribute PROFESSION.members is a set-valued aggregation of the persons belonging to that profession. Similarly, the attribute CITY.residents is a set-valued aggregation of the persons living in that city. In that case, the OID's of persons can be in their city order, or their profession order, but not both. Consequently, if a sub-object class participates in more than one type of relationship, then **at most one of these relationships will have a clustered traversal of the OID-index -- for all others, the OID's of the sub-objects will be drawn at random.**

In that case, the expected number of index pages touched at a level $i, 0 \leq i \leq H$ is given by:

$$\text{CostIndex}(i) = \text{yao}(N, B^i, s)$$

where yao(n ,m ,k) represents the expected number of pages touched when k records are drawn randomly from n records spread uniformly over m pages [YAO77]. Here B^i is the number of pages at the ith level. The number of data pages touched is given by:

$$\text{CostData} = \text{yao}(N, \frac{N}{M}, s)$$

The total cost of materialization using an OID index is:

$$\text{Cost} = \sum_{i=0}^{H} \text{yao}(N, B^i, s) + \text{yao}(N, \frac{N}{M}, s)$$

It can be shown that if $s \ll N$, then Cost is a linear function of S.

In contrast to OID representation, the materialization costs in procedural representation depend on the structure of the procedures. For example, procedures can be used to simulate OID-lists, albeit with a somewhat extra space usage. Thus, the procedure for the Toy department members might well be:

```
retrieve (employee.all) where
employee.OID = 2314 or employee.OID = 4562 or employee.OID = 9879 ...
```

In fact, if these procedures are special cased (as in parameterized procedures in POSTGRES), it is possible to factor out the extra space and thus attain a space usage which is no worse than that in OID representation. The materialization costs of these procedures are similar to those of the corresponding OID-lists.

In general, however, procedures may be totally "intensional" (where no sub-object identity is explicitly stated), totally "extensional" (as above), or anywhere in between. Thus one might choose to have the following procedure for the Toy department:

```
retrieve (employee.all) where
(employee.OID = 2314 or employee.OID = 9879 or ....) or
(employee.dept = "Toy" and employee.age < 40)
```

where the first disjunction is an extensional representation of a sub-set of employees, and the second is an intensional representation of the rest. It is clear that if the two sub-sets are disjoint, then the cost of materializing this procedure is simply the sum of the costs of its two parts. The cost of the intensional component depends on many factors like joins, indices etc. and is, in general, difficult to model.

It is however easy to determine a lower bound on the cost of materialization for procedures not involving joins. The ideal form of such a procedure is:

```
retrieve (rel.all) where
lowval <= rel.attr <= highval
```

with a proviso that an index exists on rel.attr. Note that this is possible even when a sub-object class participates in more than one relationship. In the example above, this requires an index on PERSON.city and PERSON.profession. On the other hand, since there is only one OID order possible, clustered index traversal cannot be achieved along both dimensions in OID representation. From this we can deduce that **in general, the probability of clustered index traversal in the procedural representation is higher than in the OID representation because of the higher degrees of freedom.**

Under these assumptions, the cost of this procedure is given by the following components:

$$\text{CostIndex}(i) = \begin{cases} 1 & 0 \le i < H \\ \left\lceil sB^H/N \right\rceil & i = H \end{cases}$$

$$\text{CostData} = \text{yao}(N, \frac{N}{M}, s)$$

(This assumes that the leaf pages of the index are chained together. CostData is identical for OID and procedural representations.)

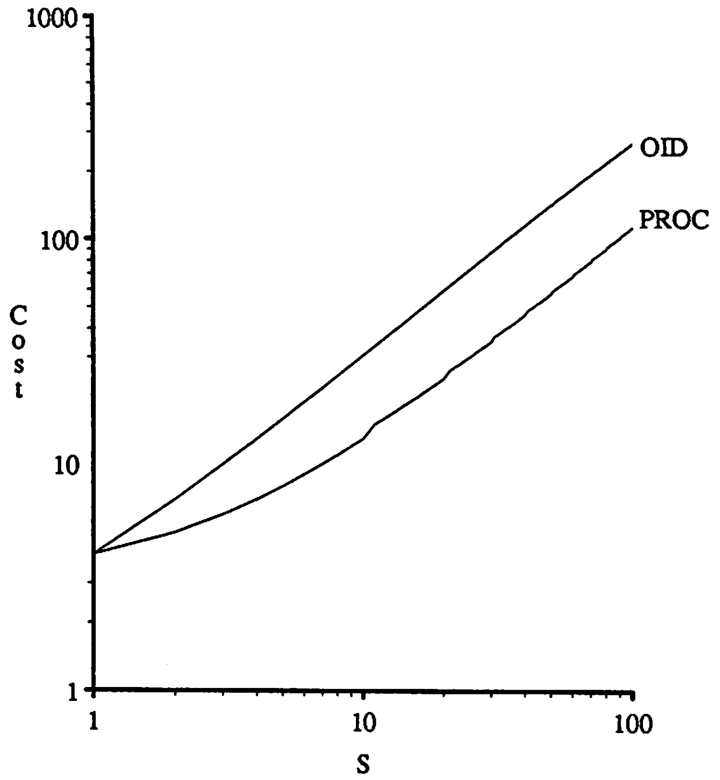

Figure 1: Cost of Materialization as a function of the number of sub-objects

Figure 1 compares the cost of materializations in the two representations. The cost difference reflects the fact that the path from the root to the leaf level of the index has to be traversed several times in OID representation, but only once in procedural representation.

In general, if a procedure on a single relation can be expressed as a disjunction containing less than s disjoint clauses, then its cost of materialization will be less than that of the corresponding OID-list. If this is not the case, then the query processor should cache in an object the OID's of the sub-objects returned by materialization of the procedure(s) in that object. Subsequent requests for materialization could use this equivalent OID-based representation. For example, if the procedure for computing the employees of the Toy department is more expensive than the corresponding OID-list, then caching the latter will help:

name	manager	location	employees
Toy	John	Denver	retrieve (employee.all) where employee.dept = "Toy" ------------------------- 2314, 4562, 9874, ...

It is shown in [JHIN90b] that the overhead for maintaining these cached OID's is minimal under most circumstances, and that the performance benefit from these rival the basic OID-based representation. Consequently, we see that under only modest space requirements, materialization in procedures costs **no more** than in the OID-based representation.

However, irrespective of the choice of primary representation, in the absence of physical clustering of sub-objects, a a major component of the total cost will still be CostData = s. Both clustering and caching alleviate this problem by reducing this component to the minimum possible. or less.

3. Clustering

If sub-objects are physically clustered with the objects that reference them, then the cost of materialization can be reduced to a minimum [BANE86]. Typically, this clustering can be achieved irrespective of the primary representation used. However, many consider it to be one of the major advantages of object-oriented database systems over traditional relational systems [BANE87]. In this section we will argue that the advantages of clustering are limited. Consequently, the claimed ease of clustering in object-oriented database systems (and hence, in OID representation) is of limited use. In [JHIN90a] we arrived at similar results using simulation techniques, but here we employ an analytic approach.

In this paper we only consider two-level hierarchies; the analysis of multiple-level hierarchies is more complicated. In a two-level hierarchy, the relationship between objects and sub-objects can be shown as a bi-partite graph with two sets of nodes, {objects} and {sub-objects}. Figure 2 gives an example of such a "relationship" (or "assignment" graph). We use the following notation:

o_i: the ith object
so_j: the jth sub-object

The set of the sub-objects of an object o_i is formally defined as:

$$SO_i = \{so_j \mid so_j \rightarrow o_i\}$$

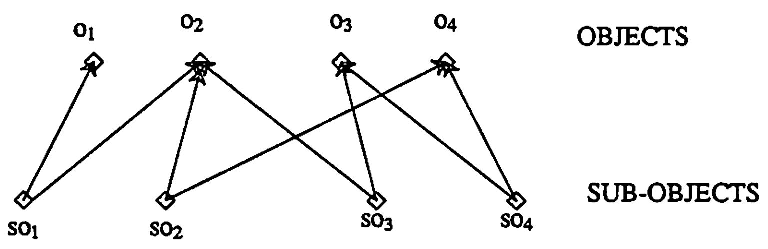

Figure 2: An Assignment Graph

where i→j refers to an edge from i to j in the graph. Similarly, the set of objects that reference a given subobject so_j is formally defined to be:

$$O_j = \{o_i \mid so_j \to o_i\}$$

We consider the following parameters:

O: Number of objects in the graph.
S: Number of sub-objects in the graph.
OF: OverlapFactor, i.e. the average number of objects sharing a sub-object.
p: Size of an object
b: Size of a bucket

We assume that the size of a sub-object is one unit. From the above parameters, it is easy to derive the following:

s: the average size of an OID-list = $\frac{S \times OF}{O}$

This is derived from the fact that the number of edges leaving {sub-objects} equals the number of edges entering {objects}.

A clustering assignment of a graph G is basically an assignment of sub-objects and objects to buckets (numbered 1 through T) with the constraint that the total weight of the objects and sub-objects assigned to any bucket does not exceed b.

The following is a generic formulation of the "goodness" of clustering, irrespective of the graph structure describing the relationships. Let BS_k and BO_k be the set of sub-objects and objects, respectively, assigned to the kth bucket. Without loss of generality, the sub-objects are in the first t ($t \leq T$) buckets. We define two sets for each bucket k. The first is the set of objects that reference the sub-objects assigned to a bucket. Formally,

$$B_k^1 = \bigcup_{so_j \in BS_k} O_j$$

The second set contains those objects assigned to k that also have at least one of their sub-objects assigned to k. Formally,

$$B_k^2 = B_k^1 \cap BO_k$$

The cost of accessing the sub-objects of an object o_i is precisely the number of buckets (excluding the bucket containing o_i) that contain at least one subobject of o_i. Assuming that all objects are accessed with the same frequency[2] and all accesses require the entire complex object, then the following formulation exists for the the cost of a clustering assignment:

$$C = \sum_{k=1}^{t} |B_k^1 - B_k^2|$$

Since $B_k^2 \subseteq B_k^1$, we have

$$C = \sum_{k=1}^{t} |B_k^1| - \sum_{k=1}^{t} |B_k^2| \qquad (3)$$

This is expressed as

$$C = B^1 - B^2 \qquad (4)$$

where B^1 and B^2 refer to the first and the second terms, respectively, in (3).

In general, a good clustering assignment will try to maximize the second term. This in turn means that the objects are packed as "densely" as possible, i.e. in the sense that there is no space in the t buckets to accommodate any more object (provided there are some objects that are assigned to buckets numbered t+1 or more)[3]. In that case, let f be the average fraction of the bucket space (for buckets numbered t or less) that is occupied by the sub-objects. Then the number of objects that can be accommodated in the buckets numbered t or less is given by

$$t\left\lfloor \frac{b(1-f)}{p} \right\rfloor$$

Under the dense packing condition, the **actual number** of objects accommodated in the first t buckets is either this quantity, or O if this is greater than O. Thus,

[2]If there is an 80-20 rule, then we just partition the database into two and apply the analysis to each part separately.

[3]This dense packing is possible if certain conditions for matching on bi-partite graphs are satisfied.

$$B^2 = \min\{O, t\left\lfloor \frac{b(1-f)}{p} \right\rfloor\} \quad (5)$$

Of course,

$$t = \left\lceil \frac{S}{bf} \right\rceil$$

We study the cost of the optimal clustering for an assignment graph G as a function of f. In that case, we rewrite (4) as

$$C(G,f) = B^1(G,f) - B^2(G,f) \quad (6)$$

Consider two graphs G_1 and G_2 which have identical values of p, O and S. Furthermore, let b be the same for the optimal clustering assignments of both. In that case,

$$B^2(G_1,f) = B^2(G_2,f) = B^2(f)$$

In Figure 3, we have plotted $B^1(G_1,f)$, $B^1(G_2,f)$ and $B^2(f)$. Figure 4 plots $C(G_1,f)$ and $C(G_2,f)$. While $C(G_1,f)$ has a minima at $f = 0.3$, $C(G_2,f)$ is a monotonically decreasing function and hence has a minima at $f = 1$. It is clear that for a graph, if the slope of $B^1(f)$ component of the cost of a clustering assignment is

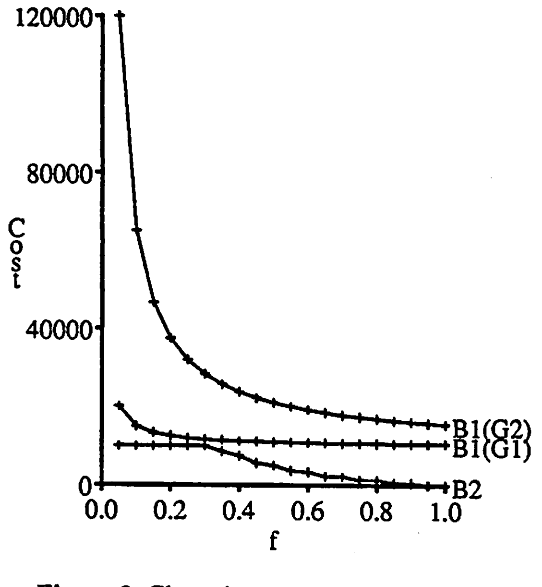

Figure 3: Clustering cost components for two graphs

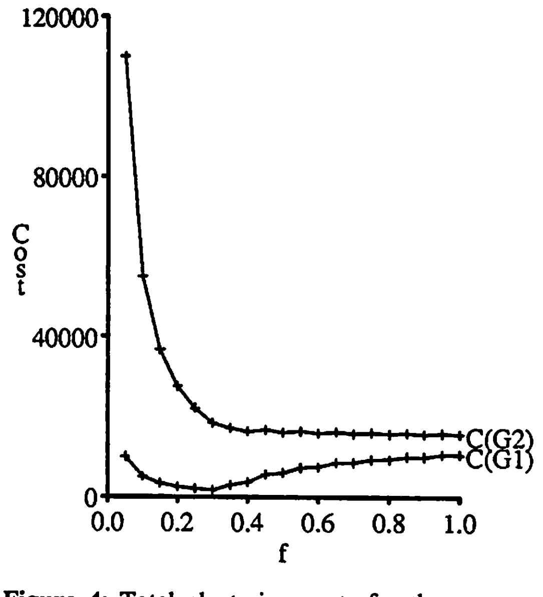

Figure 4: Total clustering costs for the two graphs

strictly less than (i.e. more negative, and consequently steeper than) that of its $B^2(f)$ component, then the minima is going to occur at $f = 1$. Otherwise, there exists a value of f where C(f) is at its minimum. A value of $f < 1$ implies some objects are clustered with their sub-objects. When $f = 1$, the objects and sub-objects are clustered separately (as perhaps separate relations). Thus from Figure 4, clustering makes sense for the first graph, but not for the second graph.

Before studying clustering performance of general assignment graphs, we first study the properties of **perfectly regular** graphs. These graphs have a simple description for their optimal clustering assignments.

3.1. Optimal Clustering for Perfectly Regular Graphs

For a given set of parameters, O, S and OF, a perfectly regular graph G_P is defined to be the graph with the following edges:

$$\text{for all } i,\ o_i \leftarrow so_j,\ i\mu \le j \le (i\mu + s - 1) \bmod S$$

where $\mu = \frac{S}{OF}$ is the largest size of a block of sub-objects that belong to the same set of objects. In Figure 5 we show the adjacency matrix for our perfect graphs. The rectangles represent 1's and the rest of the matrix contains 0's. A 1 in the position (i, j) implies that the sub-object so_j is assigned to the object o_i. For

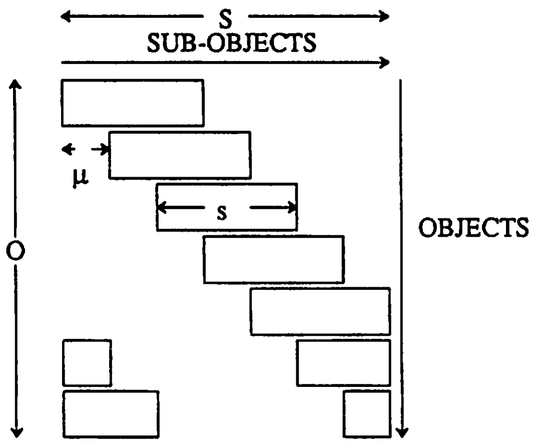

Figure 5: The adjacency matrix for a perfectly regular graph with shown parameters

a given value of f, it can be shown that in (one) optimal clustering, the sub-object so_j is assigned to the bucket $\frac{j}{bf}$. For the kth bucket, it can be shown that $|B_k^1|$ (the number of objects that reference one or more of the sub-objects assigned to k) is:

$$B_k^1 = \left\lfloor \frac{(kbf + bf - 1)}{\mu} \right\rfloor - \left\lceil \frac{(kbf - s + 1)}{\mu} \right\rceil + 1$$

and hence B^1 is easy to compute. Consequently, we can plot C(f) and hence determine $f = f_{min}$ where C(f) is minimum.

In Figure 6 we have plotted $\frac{C(f=f_{min})}{C(f=1)}$ as a function of OF. If this ratio is less than one, then $f_{min} < 1$, and consequently, clustering is beneficial. In one of the graphs (Vary_s) we have kept O and S constant (both at 10000); consequently s (the size of an OID-list) increases with OF (recall that $O \times s = S \times OF$). For the other two graphs, O was increased with increasing OF in such a way that s remained

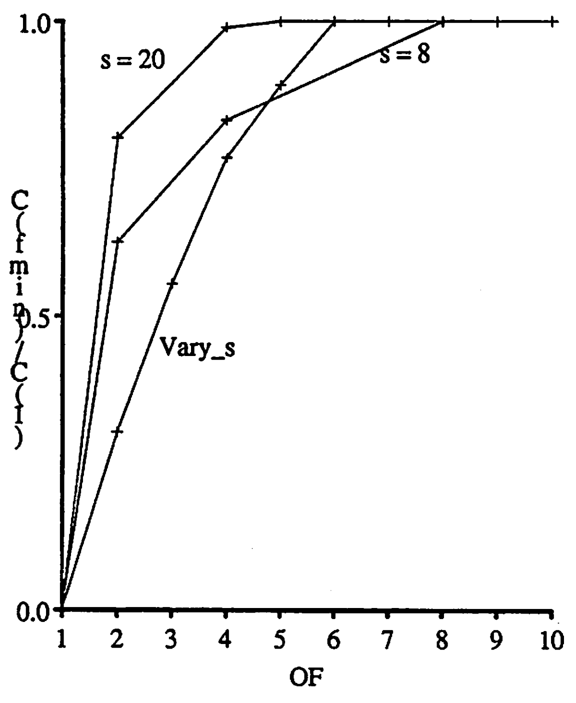

Figure 6: Clustering Performance for Perfectly Regular Graphs

constant. In one graph, s was fixed at 8, and in the other, at 20. The size of the bucket was fixed at 10 units and that of an object at 2 units.

We see from all the curves that when OF = 1, clustering is ideal. This is to be expected because if sub-objects are not shared, they are best clustered with the only object that references them. However, by doing so, we will in effect have simulated a value-based representation. Consequently, when OF = 1 and sub-objects are to be clustered with the objects, OID representation does not make much sense.

When a sub-object is shared by two to seven objects, clustering is likely to help. However, the cost of an optimal clustering assignment fast approaches that of no clustering, indicating that the benefits of clustering are only marginal in this region. When eight or more objects share a sub-object, clustering is counter-productive, and it is best to keep the objects and sub-objects separately.

The other factor affecting clustering performance is s. If the set of sub-objects of an object is large, clustering may not help fit all the sub-objects into one bucket. Simultaneously, this will also decrease the space available to accommodate other objects that reference these sub-objects. Consequently, clustering is likely to fail for large s, and this is exactly what is observed. The curve for s = 20 rises much more sharply than for s = 8. There is a specific reason why larger values of s are interesting. Clustering in multiple-level hierarchies can often be viewed as recursive two-level clustering, starting bottom-up. In these cases, by the time we reach the second or the third level from the bottom, the net value of s would be quite large. Thus we feel that clustering multiple-level hierarchies is not likely to be very beneficial.

In conclusion, clustering is viable for perfectly regular assignment graphs provided the size of the OID-lists (s) and/or the number of objects sharing a sub-object (OF) is small (generally less than 6 or 7).

3.2. Optimal Clustering for Random Graphs

Optimal clustering assignment for random graphs is a very hard problem [KERN70]. However, as we show, it is possible to study the properties of optimal clustering without giving a solution for the same.

It is clear that the assignment of sub-objects to buckets is uniquely identified by a permutation π of the sub-objects such that the sub-object π_j is assigned to the bucket $\frac{j}{bf}$ (assuming each bucket contains the same number of sub-objects). For a permutation π, we define a "covering" to be the set of rectangles of the form R_a^i where a is the maximal set of contiguous (modulo S) sub-objects that are referenced by an object

o_i. In a perfectly regular graph, there exists only one rectangle for each object, whereas in general, this is not the case. For example, the covering for the permutation $\{so_2, so_3, so_1, so_4\}$ for the assignment graph in Figure 2 is given in Figure 7. For such a covering, we define μ to be the average distance between two contiguous starting points of the constituent rectangles. For example, in the above figure, $\mu = 0.8$. The number of rectangles in the covering then is $NR = \frac{S}{\mu}$. By the definition of the covering, $NR \geq O$. For the perfectly regular graph there exists the obvious permutation where $NR = O$.

Assuming that the number of buckets is large enough, the probability of existence of two rectangles of the type R_a^i and R_b^j where some sub-object from a falls into the same bucket as a sub-object from b is extremely low. It can then be shown that

$$B^1(G_R, f) = \sum_{k=1}^{t} NR_k = t \times \overline{NR}$$

where NR_k is the number of rectangles with sub-objects in the bucket k.

The closed-form for $\overline{NR}$ is quite complicated. If the number of buckets is large enough, then a statistical estimate for this can be obtained experimentally. In Figure 8 we have plotted $B^1(\mu, f)$ where a generic graph is identified by its μ value. For the perfect graph, $\mu = \frac{S}{OF}$. The applicable parameters are: $O = 10000$, $S = 10000$ and $OF = 3$

When $f = \frac{1}{b}$, all the curves have identical values. This is because if only one sub-object is to be stored in a bucket, then $NR_k = OF$ and consequently, $B^1 = t \times OF = S \times OF$ (which is independent of μ). The curves start deviating with increasing f, and the smaller the μ, the larger the deviation from the curve of the

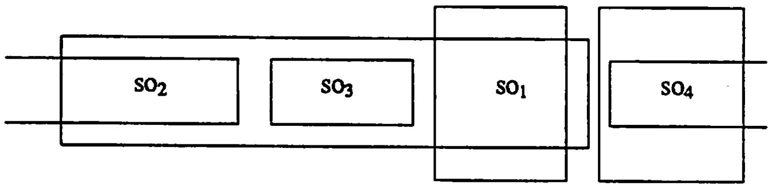

Figure 7: Covering for a Permutation for the Graph in Figure 2

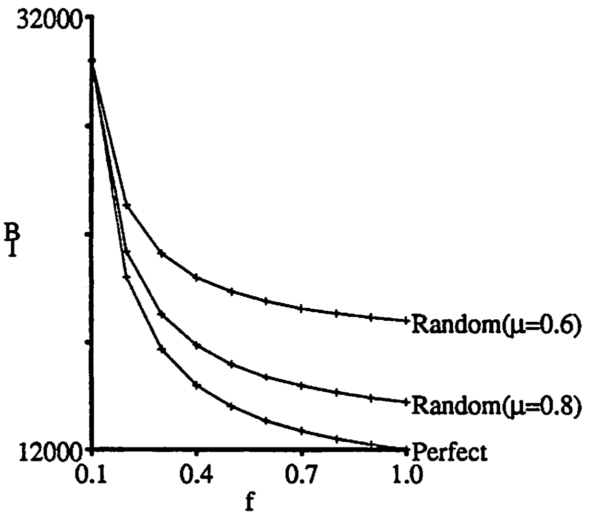

Figure 8: Clustering cost components for graphs with varying μ

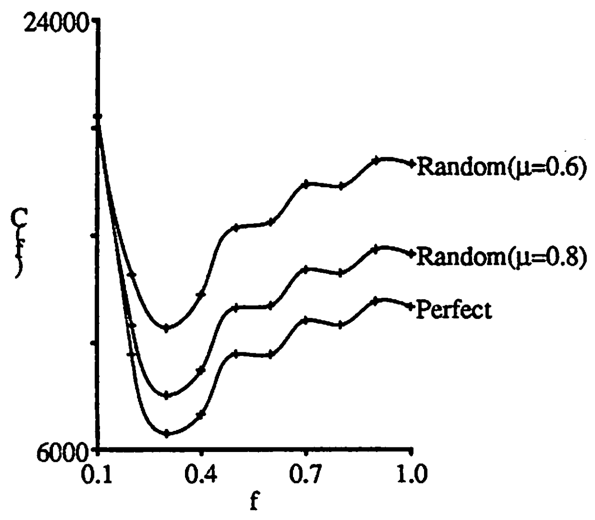

Figure 9: Total clustering costs for graphs with varying μ

perfectly regular graph. In spite of these deviations, the three graphs have identical values of f_{min} (Figure 9).

From a broad range of experiments, it was observed that the minima of C(f) for a graph was a function of O, S and OF, and in general, independent of μ. Consequently, the behavior of a random graph can be characterized by that of a perfectly regular graph having the same set of parameters, and hence is helped by clustering only when the size of its OID-lists and the OverlapFactor are small.

There are a couple of other reasons why clustering is not that attractive. The first is the obvious difficulty in achieving a good clustering; it is computationally intensive and does not work when the objects and sub-objects are being dynamically added and deleted. In [JHIN90a] we have established another factor which prevents clustering from being the optimal choice. To facilitate merge based join processing of large queries, it is sometimes important to have the entities in the database sorted in OID order. When clustering is employed, it is difficult to ensure this ordering because then OID's become location dependent.

We end this section with a brief discussion on another aspect of sharing. In procedural representation, for example, sharing is possible if 1) Two procedures return some common tuples, or 2) Two objects have **identical** procedures. Similarly, in OID representation, two objects might have intersecting OID-lists, or identical ones. Till now we have been discussing only the first model of sharing. To accommodate the second, we need to introduce an intermediate layer between the objects and sub-objects in Figure 2. This layer (of procedures or OID-lists) receives edges from {sub-objects} and has edges into {objects}. The average number of edges leaving a node at this level is termed ShareFactor, or SF for short. If SF = 1, then in effect we have the first model of sharing.

4. Separate Caching

In the absence of clustering, accessing the sub-objects of an object typically involves multiple data page I/O's (as is evident from the cost model we have developed in Section 2). Caching the materialized result can reduce this cost appreciably. This caching can be in two places. In **inside caching**, the materialized result is stored with the object that contains the appropriate procedure (or OID-list). In contrast, in **separate caching**, all cached results are stored in one place, separate from the objects. It is shown in [JHIN88, SHEK89] that if SF exceeds a small constant, then separate caching is the better option. It entails lesser space usage, and fewer updates, but at the expense of one more I/O to retrieve the cached result [JHIN88]. Therefore in this paper we only examine separate caching for procedural and OID representation.

The flip side of caching is that updates to sub-objects entail either invalidating or updating the cached values. Consequently, caching is unattractive in presence of high update traffic. In what follows, when we talk of caching procedures and OID-lists, we mean caching the results of appropriate materializations.

A cached value needs to be accessed from two directions:

1) From the side of the objects containing the corresponding procedure or OID-list because it is needed to answer queries on that complex object.

2) From the side sub-objects because we may need to invalidate (or update) a cached value because of an update to a sub-object.

We need to be able to efficiently do both for separate caching to work. For 2) to work, we must store some pointer in (at least) those sub-objects whose value determines the cached result. This pointer should either directly or indirectly be able to give the location of the corresponding cached result. We term these pointers as **I-locks** (short for invalidation locks). These locks are held on (at least) the sub-objects that constitute a cached result. An exclusive (write) lock acquired on a sub-object containing an I-lock necessitates some write action to the corresponding cached result. While we could have sophisticated schemes that update the cached result to maintain its currency, in this paper we assume a simple scheme which invalidates the corresponding cached result. We further assume that these I-locks remain in place even when their cached results are invalidated -- consequently I-locks have to be installed only when a result is first materialized and cached. Subsequently, we need to install I-locks only in those sub-objects that are added to this object since the first materialization. Furthermore, we need to communicate the possible location of a cached result to all the objects that can share it (i.e. they have the same procedure, or the same OID-list).

In procedural representation, subsequent addition of sub-objects is handled through one of the many predicate locking schemes. A simple scheme, called Segmented B-Trees, inserts locks in an appropriate index at the highest level possible [KOLO89]. I-locks inherited by newly added sub-objects are easily discovered from the corresponding I-locks on the indices. This locking scheme has a modest space overhead, but little performance penalty. In OID-based representation, predicate level locking is not needed since references to newly added sub-objects are explictly added to the existing objects.

Under the reasonable assumption that objects that can share a cached result do not know about each other, we need a mechanism that lets all these objects independently compute the location of where the result might be cached. In procedural representation, this means that the location must be a hash function of the procedure body, and in OID-based representation the location must be a hash function of the OID-list. Consequently if two objects O1 and O2 share a procedure P (or equivalently, an OID-list L), then if the result is cached during the processing of a query against O1, it can be used for answering queries against O2.

This scheme causes problems for OID-based representation because the location of a cached result is no longer guaranteed to be a function of the I-locks. For example, if the result of an OID-list L = {s1, s2} is first cached, I-locks on s1 and s2 reflect the location which is a function of L. Let us say that subsequently a sub-object s3 is added to L, making L' = {s1, s2, s3}. In that case, I-locks on s1 and s2 are totally incorrect about the location of the cached result which happens to be a function of L'.

There are many possible solutions to this problem, including removing I-locks whenever a cached result is marked invalid, and setting them every time a result is cached (consequently, I-locks always reflect the latest location). This turns out to be fairly expensive. Instead, a simpler solution which involves forcing an identity on an OID-list which is independent of its constituent sub-objects is suggested in [JHIN90b]. In this, the location (identity) of a cached result is determined the first time it is cached. This location is inviolate for the life of the database and all the objects that can share this result must learn this location (identity). This in turn involves building an index on the cached values which maps the OID-list as an object knows it, to its identity. Maintaining this index, and traversing it to determine the identity of a given OID-list is expensive. Details of this scheme can be found in [JHIN90b].

[SHEK89] suggests a scheme for separate caching in OID representation. However, it makes no mention of how it handles insertions and deletions of sub-objects. As we have shown, it is these operations that cause the real problems in an OID based representation.

4.1. Performance Considerations

Even without separate caching, it is clear that OID representation will be outperformed by procedural representation because in the latter, insertions or deletions of sub-objects **do not cause updates to objects.** If a sub-object is shared on the average by SF×OF objects, then every addition/deletion of sub-object entails SF×OF writes on the average. If the definition of an OID-list is kept separately, then this write activity can be reduced to OF, but at the expense of an indirection to fetch the definition.[4]

In order to quantify the above discussion, we performed a simulation study using Commercial INGRES [RTI86]. Objects with similar attributes were stored in a relation, and the cached results were stored in a separate relation. At any time, cache did not contain more than 1000 valid values. The total number procedures/OID-lists in the database was set at 5000. Modifications to objects and sub-objects (including insertions and deletions) were done at frequencies depending on certain parameters. Here we present only one result, the rest of the details can be found in [JHIN90b]. Figure 10 plots the cost of

[4] In [JHIN90b] we establish that even if the cost of materialization in procedural representation is higher than that in OID based representation, a sufficiently high activity to the set of sub-objects constituting a complex object results in procedural representation outperforming the OID-based representation.

separate caching for procedures and OID's as a function of the frequency at which sub-objects are inserted into the database. It is clear that caching in both OID and procedures deteriorates with an increased frequency of insertions (and deletions) of sub-objects. This is to be expected since an increase in the number of insertions (deletions) of sub-objects results in more invalidations of cached values; and consequently in a higher cost.

However, as we see, OID caching 1) performs worse than procedure caching, and 2) deteriorates faster than procedure caching. To explain these phenomena, we separate the cost difference into two components: from the insertions/deletions of objects, and from insertions/deletions of sub-objects.

In OID-based representation, newly added objects must traverse an index (as mentioned before) to determine the identities of their results. Since we are maintaining at a constant the frequency of insertions

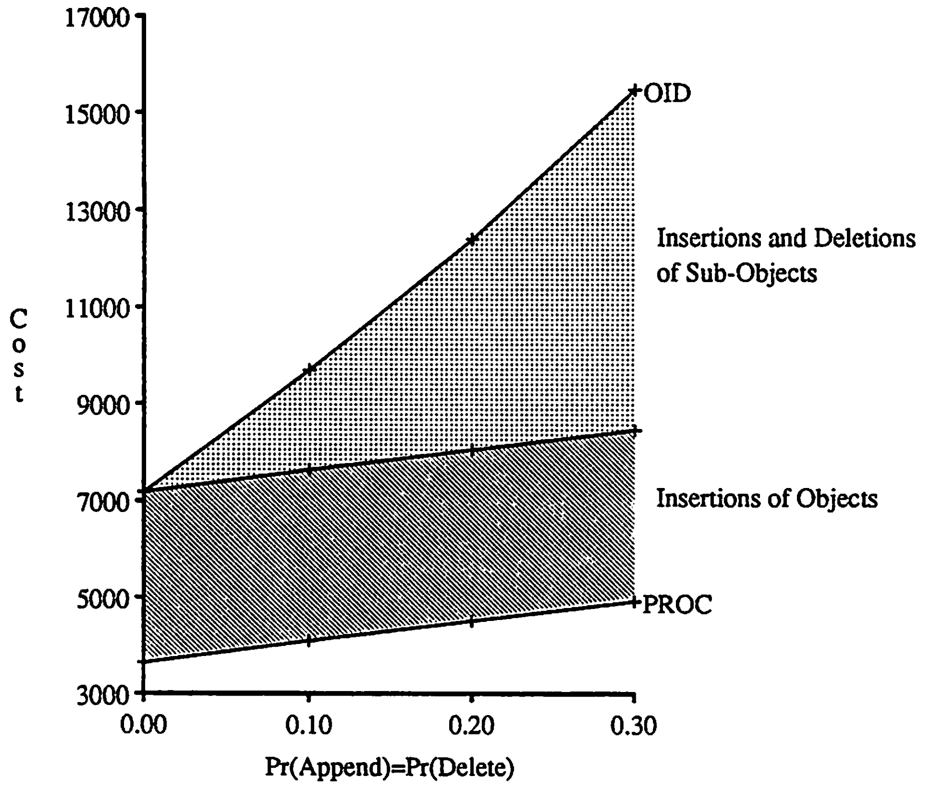

Figure 10: Separate Caching as a function of Append Frequency for Sub-Objects

of objects, this component of the difference remains at a constant.

The second component of the difference reflects the fact that on each insertion/deletion of the sub-objects, the definition of the corresponding OID-lists (expected OF in number) must be modified, along with setting the invalid bit for those results that are cached. In procedural representation, the only cost is to invalidate the cached results. Hence if a fraction f of the possible OID-lists are cached, then for every insertion (deletion), OID representation will do $\frac{1-f}{f}$ times more work than procedural representation. As the frequency of insertions (deletions) goes up, this represents larger and larger difference.

In summary, separate caching is not very viable for OID representation. However, the two representations perform similarly on inside caching. Thus caching works for OID representation only if SF is low. For procedural representation, some form of caching is useful, regardless of SF.

5. Query Modification

In the presence of certain special class of procedures (termed **parameterized**) procedures, queries on complex object can often be processed using query modification. For example, consider the following schema in POSTGRES:

```
ORG (name, city, activity)
SCIENTIST (name, city, membership)

where SCIENTIST.membership contains procedures of the form:
        retrieve (ORG.all) where
        ORG.city = $.city and
        ORG.type = "Professional"
```

signifying the fact that a complex object of type "SCIENTIST" is made of sub-objects from "ORG" (organizations), and that these sub-objects can be evaluated using the procedure above. ($ refers to the corresponding tuple of "SCIENTIST". The new syntax of procedures in POSTGRES, Version 2 is slightly different [STON89].) Consequently, a query of the form:

```
retrieve (SCIENTIST.name) where
SCIENTIST.membership.size > 50
```

(i.e. the scientists who belong to a large organization) can be processed by rewriting to:

```
retrieve (SCIENTIST.name) where
SCIENTIST.city = ORG.city and
ORG.type = "Professional" and
```

ORG.size > 50

If the original query is evaluated without re-writing, then it must be processed top-down. Under some circumstances, a bottom-up processing (i.e. retricting ORG and then finding the matching tuples of SCIENTIST) might be less expensive. In such a bottom-up processing, in procedural representation, either of the two selections on ORG can be used to restrict it. In contrast, in OID representation, only the last clause will be available for restriction. Consequently, the number of I/O's required to restrict the sub-objects in procedures is never more than that in OID's. This is a reflection of the fact that more semantic information can be encoded in the procedures compared to OID's.

The exact detail on the performance implication of these differences can be found in [JHIN90b].

6. Conclusions

Procedural and OID representation represent two different approaches for modeling the relationships between objects. The former uses an intensional representation, whereas the latter explicitly lists the identifiers of the sub-objects (in the form of an OID-list) of an object.

In this paper we compared the two representations on four major axes. Table 2 shows how the two representations perform on these dimensions. The first axis is the cost incurred to determine the values of the sub-objects of an object. In procedural representation, this is the cost of evaluating a procedure. In OID representation, this is the cost of dereferencing the OID's in a list to get to the actual location of its sub-objects. A formal cost model for materialization in OID representation and in certain special cases of procedural representation was developed. This cost model was used to show that single relation intensional procedures will be in general cheaper than OID-lists. Furthermore, a simple scheme of caching the OID's of expensive procedures ensures that procedures are no more expensive than OID-lists.

Since these materialization costs are the bottleneck in evaluating queries on complex objects, we next looked at three techniques used to mitigate this effect. The first mechanism examined was clustering. In order to determine the performance benefit from clustering sub-objects with the objects that reference them, we studied analytically the cost of an optimally clustered "perfectly regular" assignment graph. It was shown that clustering on two-level hierarchies is beneficial, provided the sub-objects are not shared by more than a small number (say 5-7) objects, and that the number of sub-objects of an object is similarly

Dimension	Representation	
	OID	Procedural
Materialization	Approximately linear in number of sub-objects	Relatively inexpensive on single relations, varying otherwise. Caching OID's make it always cheaper than OID-based
Clustering	Wins if OF and $s < 5$ to 7. Wins big if $OF = s = 1$	If tuples from different relations can be clustered on the same bucket, then same as OID. Else, loses big when $OF = 1$, marginally loses for s and OF between 2 and 7, and performs similar to OID otherwise.
Caching	Wins unless heavy update (especially insertions and deletions) traffic to sub-objects	Wins big, since comparatively little effect of insertions and deletions to sub-objects
Query Modification	N.A.	Wins on parameterized procedures

Table 2: A Summary of the Performance Comparison

small. Results for random graphs were then extended from those of perfect graphs and similar conclusions were reached. Hence even if it is assumed that clustering is one of the major advantages that object-oriented (and hence OID based) systems offer over relational systems, the performance gains from such a difference are rather limited.

A major difference between the two representations is the way they treat additions and deletions of sub-objects. Procedural representation outperforms OID-based representation when this update traffic is heavy. This effect mainfests itself when we study separate caching too. It was shown that separate caching in OID's is at an inherent disadvantage because of the lack of an "identity" for the OID-lists that is invariant with the additions and deletions of sub-objects. A solution to this problem was then presented and the performance results of this approach showed that procedural representation may outperform OID representation by 50% or more.

We also had a brief look at another differentiating yard-stick -- query modification. Queries on parameterized procedures can be flattened, and these flattened queries often perform better in procedural representation.

In summary, the choice of representing complex objects should lean towards procedural representation, provided the following two criteria are satisfied: 1) The procedures should be intensional in nature, and 2) The number of sub-objects of an object and/or the number of objects that share a sub-object should be greater than a small constant.

References

[BANE86] Banerjee, J. and Kim, W., "Clustering a DAG for CAD Databases," MCC Technical Report Number: DB-128-85, Microelectronics and Computer Technology Corporation, Feb. 1986.

[BANE87] Banerjee, J., et al., "Data Model Issues for Object-Oriented Application," ACM Trans. on Office Info. Sys. 5(1), Jan. 1987.

[BATO85] Batory, D.S., and Kim, W., "Modeling Concepts for VLSI CAD Objects," ACM Trans. on Database Systems, 10(3), Sept. 1985.

[CARE88] Carey, M. et al., "A Data Model and Query Language for EXODUS," Proc. ACM-SIGMOD Conf., 1988.

[COPE84] Copeland, G. and Maier, D., "Making Smalltalk a Database System," Proc. ACM-SIGMOD, 1985.

[COPE85] Copeland, G.P. and Khoshafian, S.N., "A Decomposition Storage Model," Proc. ACM-SIGMOD, 1985.

[DADA86] Dadam, P. et al., "A DBMS Prototype to Support Extended NF^2 Relations: An Integrated View on Flat Tables and Hierarchies," Proc. ACM-SIGMOD, 1986.

[JHIN88] Jhingran, A., "A Performance Study of Query Optimization Algorithms on a Database System Supporting Procedures," Proc. VLDB, 1988.

[JHIN90a] Jhingran, A., "Alternatives in Complex Object Representation: A Performance Perspective," Proceedings, Sixth International Conference on Data Engineering, 1990.

[JHIN90b] Jhingran, A., "On Alternatives in Complex Object Representation: A Performance Perspective," PhD Thesis, in preparation.

[KERN70] Kernighan, B.W. and Lin, S., "An efficient heuristic procedure for partitioning graphs," Bell Systems Technical Journal 49, 1970.

[KHOS86] Khoshafian, S.N. and Copeland, G.P., "Object Identity," Proc. of OOPSLA, 1986.

[KIM87] Kim, W. et al., "Operations and Implementation of Complex Objects," Proc. Conf. on Data Engr., 1987.

[KOLO89] Kolovson, C. and Stonebraker, M., "Segmented Search Trees and their Application to Data Bases," (in preparation).

[LORI85] Lorie, R. et al., "Supporting Complex Objects in a Relational System for Engineering Databases," in Query Processing in Database Systems, eds. Kim, W., Reiner, D. and Batory, D., Springer-Verlag, 1985.

[ROWE87] Rowe, L. and Stonebraker, M., "The POSTGRES Data Model," Proc. VLDB, 1987.

[RTI86] Relational Technology Inc. INGRES Release 5.0 Reference Manuals, 1986.

[SHEK89] Shekita, E.J. and Carey, M.J., "Performance Enhancement Through Replication in an Object-Oriented DBMS," Proc. ACM-SIGMOD, June 1989.

[SMIT77] Smith, J.M. and Smith, D.C.P, "Database Abstractions: Aggregation and Generalization," ACM Trans. on Database Sys., 2(2), June 1977.

[STON86] Stonebraker, M. and Rowe, L., "Design of POSTGRES," Proc. ACM-SIGMOD, 1986.

[STON89] Stonebraker, M. et al., "On Rules, Procedures, Caching and Views in Database Systems," Tech. Report UCB/ERL Memo M89/119, University of California, Berkeley, Oct. 1989.

[VALD86] Valduriez, P. et al., "Implementation Techniques for Complex Objects," Proc. VLDB 1986.

[YAO77] Yao, S.B., "Approximating Block Accesses in Database Organizations," Communication of the ACM, 20(4), Aug. 1977.

[YAO85] Yao, S.B., ed. "Principles of Database Design," Prentice Hall Inc., 1985.

[ZANI85] Zaniolo, C., "The Representation and Deductive Retrieval of Complex Objects," Proc. VLDB, 1985.