Query processing remains one of the important challenges of Object-Oriented Database Management Systems. Cost based query optimization involves creating alternative executing plans for a given query and executing the least costly one within a cost model framework.

In Object-Oriented Database Management Systems (OODBMSs) objects may store references to other objects (precomputed joins), and therefore path expressions are used in query languages. The cost formulas for explicit joins and the selectivities of attributes and joins are well-known in the relational model and there is a need for a similar work for object-oriented queries involving path expressions. This information is necessary for deciding whether to use pointer chasing or to convert the path expressions into explicit joins and also for deciding on the execution order of path expressions. In this paper, we provide a cost model that includes the formulas for the costs and selectivities of forward and backward path traversals.

The goal of query optimization is to find an execution plan for a specific query in order to minimize the cost of executing the query. The steps involved in this process can be considered at two levels, logical query optimization (query rewriting) using semantic properties of the language in order to find expressions equivalent to the one given by the user, and physical query optimization, based on a cost model to choose the best algorithm for evaluating the query.

In OODBMSs, since complex object definition involves nested objects, the query language must easily allow the specification of predicates on a nested sequence of attributes. Therefore the notation of path expressions has been introduced in the object-oriented query languages.

In calculating the cost of an execution plan for object-oriented queries, estimation of the selectivities and costs of forward and backward traversals in path expressions is necessary. Although executing precomputed (implicit) joins is not the best in all situations, estimations of the cost of path traversals are essential to compare their costs with other possible alternatives to choose the best performing access plan. The fact that path traversals must be taken into account when deciding on an execution plan is demonstrated through the following query:

```
SELECT emp.name, emp.job.name, emp.dept.name
FROM Employee emp
WHERE emp.dept.plant.location='Dallas' and emp.eno=15
```

Assuming that there is no index in any of the extensions of the classes, this query will require one sequential scan of the extension of the Employee class and 3 disk accesses, (one to fetch the corresponding dept object, one to retrieve the plant object and the last one to fetch the job object) when the path expression is evaluated through pointer chasing. If this query is executed with an explicit join technique it will require the sequential scan of the extensions of the emp, dept, plant, and job classes, and even for very small sized extensions, this cost will exceed the cost of path traversal.

On the other hand, the optimal execution plan for the following query is through explicit joins (Blakeley, McKenna & Graefe, 1993):

```
SELECT emp.name, emp.job.name, emp.dept.name
FROM Employee emp
WHERE emp.dept.plant.location='Dallas'
```
It is clear that to be able to decide on the better execution plan, we should be able to calculate the cost of possible alternatives which includes the plans for the traversal of the path expressions. Therefore, there is a need for a cost model for the path expressions.

In this paper, we provide the formulas for calculating the selectivities of path expressions, for estimating the size of an explicit join involving path expressions, and for calculating the costs of forward and backward traversals.

Previous work on object-oriented query optimization involved the definition of new object algebras (Alhajj & Arkun, 1993; Shaw & Zdonik, 1990; Straube & Ozsu, 1990), query rewriting techniques (Cluet & Delobel, 1992), and new execution algorithms to efficiently traverse complex object structures such as pointer based joins (Shekita & Carey, 1990) and complex object assembly (Keller et al., 1991). Recently the design and implementation of a query optimizer based on complete extensible framework has been reported in Blakeley et al. (1993). In Blakeley et al. (1993), only the selectivities of the attributes are considered when there is index on them that can be used to assist selectivity estimation. In other cases, selectivities are assumed to be 10%.

In Kemper & Moerkotte (1990), access support relations are introduced as a means for optimizing query processing in object-oriented database systems. The general idea is to maintain redundant separate structures (disassociated from the object representation) to store object references that are frequently traversed in database queries. Access support relations relate objects with each other as opposed to relating an object to an atomic value as in conventional indexing (Kemper & Moerkotte, 1990). In this work, an analytical cost model for access support relations and their application is developed.

In Bertino and Martino (1993), parameters of a cost model are presented by taking the class inheritance hierarchy into account. Based on these parameters, cost formulas are derived for a path expression. Since no selectivity definition for path expressions is provided, the work is restricted to queries involving a single path expression.

In Kim (1990) and Bertino (1993) nested indices and path indices are introduced for processing path expressions. A nested index establishes a direct connection between the object at the start of and the object at the end of the path instantiation. A path instantiation is a sequence of objects obtained by instantiating the classes belonging to the path. The index key is the object at the end of the path instantiation. A path index, on the other hand, store instantiations (that is, sequences of objects) of a path. The index key is the object at the end of the path instantiation, as in the nested index.

Cost Model Parameters

In the object model that Atkinson et al. (1992) used in this paper, complex objects are built from simpler ones by applying constructors to them. The simplest objects are integers, characters, byte strings of any length, Booleans, and floats. The complex object constructors are Tuple, Set, List and Reference. Any constructor can be applied to any object. Each object has a unique Object Identifier (OID). Objects are grouped in the abstraction level of a class, in other words, classes have extensions. Each object is as a member of only one class. If an object is an instance of the class A, then it will of necessity be a member of the extent of A. Finally, our object model assumes that there are no backward links.

It should be noted that object data model where any constructor applies to any object is very much different from the relational data model where the set constructor can only be applied to tuples and the tuple constructor can only be applied to atomic values. For this reason the cost model parameters of the relational model are inadequate for the object model; new cost model parameters need to be defined.

In this section cost model parameters are defined and calculated for the object model presented. These parameters are used in various selectivity calculations which form the basis of the cost calculation of path traversals. In Table 1, the cost model parameters are presented. Similar cost model parameters have been defined in Kemper and Moerkotte

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Short Hand Notation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1C_i$</td>
<td>-</td>
<td>Total number of instances of $C_i$</td>
</tr>
<tr>
<td>nbpages($C_i$)</td>
<td>-</td>
<td>Total number of pages $C_i$ occupies</td>
</tr>
<tr>
<td>size($C_i$)</td>
<td>-</td>
<td>Size of an instance of class $C_i$</td>
</tr>
<tr>
<td>notnull(A, $C_i$)</td>
<td>-</td>
<td>The proportion of instances of class $C_i$ where attribute A is not null</td>
</tr>
<tr>
<td>fan($A$, $C_i$)</td>
<td>fan($A$)</td>
<td>The average number of instances of class $C_i$ that are referenced by an instance of $C_i$ through attribute $A$</td>
</tr>
<tr>
<td>totref($A$, $C_i$)</td>
<td>totref($A$)</td>
<td>The total number of objects in class $C_i$ that are referenced by at least one object in class $C_i$ through attribute $A$</td>
</tr>
<tr>
<td>dist($A$, $C_i$)</td>
<td>dist($A$)</td>
<td>The number of distinct values of the atomic attribute A of class $C_i$</td>
</tr>
<tr>
<td>max($A$, $C_i$)</td>
<td>max($A$)</td>
<td>The maximum value of the atomic attribute A of class $C_i$</td>
</tr>
<tr>
<td>min($A$, $C_i$)</td>
<td>min($A$)</td>
<td>The minimum value of the atomic attribute A of class $C_i$</td>
</tr>
</tbody>
</table>

Table 1. Cost Model Parameters