A Cost Model for Path Expressions in Object-Oriented Queries

Cetin Ozkan Asuman Dogac Mehmet Altinel
Software Research and Development Center
Scientific and Technical Research Council of Turkiye
Middle East Technical University
06531, Ankara Turkiye

e-mail: asuman@vm.cc.metu.edu.tr

Abstract

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 path expressions are used in query languages. Although the cost fomulas for explicit joins and the selectivities of attributes and joins are well-known in the relational model, there is no similar work involving path expressions for OODBMSs. However in order to optimize object-oriented queries involving path expressions, a cost model is essential. This information is necessary for deciding whether to use pointer chasing or to convert the path expressions into explicit joins and also for deciding 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.

1. Introduction

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 calculating the cost of an execution plan for object-oriented queries, estimation of the cost of forward and backward traversals in path expressions is necessary. Although executing precomputed joins is not the best in all situations, estimations of the cost of path traversals is 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 [Bla 93] :

```
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. Although the cost formulas for explicit joins and selectivities of attributes, and joins are well-known in relational model, there is no similar work in calculating the selectivities and costs of path expressions for OODBMSs.

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 [Alh 93, Sha 90, Str 90], query rewriting techniques [Clu 92], and new execution algorithms to efficiently traverse complex object structures such as pointer based joins [She 90] and complex object assembly [Kel 91]. Recently the design and implementation of a query optimizer based on complete extensible framework has been reported in [Bla 93]. In [Bla 93], 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%.

2. Cost Model Parameters

In the object model [Atk 92] 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.

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 the Table 1, the cost model parameters are presented. Similar cost model parameters have been defined in [Kem 90], [Ber 92] and [Ber 93]. In defining the cost model, [Kem 90] considers the extensions of classes where in [Ber 92] and [Ber 93], the class inheritance hierarchy is also taken into account. Our cost model considers the class extensions. Furthermore we have defined some more parameters that serve our purposes better.

In the Table 1, C_i is a class, A is either an attribute or a parameterless method of class C_i with an atomic return type which is treated in the same way as an atomic attribute, and C_{i+1} is the class referenced by attribute A of class C_i.

Parameter	Short Hand Notation	Definition
$\lvert C_i \rvert$	-	Total number of instances of C_i
nbpages(C_i)	-	Total number of pages C_i occupies
size(C_i)	-	Size of an instance of class C_i
notnull(A, C_i)	-	The proportion of instances of class C_i where attribute A is not null
fan(A,C_i,C_{i+1})	fan_i(A)	The average number of instances of class C_{i+1} that are referenced by an instance of C_i through attribute A
totref(A,C_i,C_{i+1})	$totref_i$(A)	The total number of objects in class C_{i+1} that are referenced by at least one object in class C_i through attribute A
dist(A, C_i)	$dist_i$(A)	The number of distinct values of the atomic attribute A of class C_i
max(A, C_i)	max_i(A)	The maximum value of the atomic attribute A of class C_i
min(A, C_i)	min_i(A)	The mimimum value of the atomic attribute A of class C_i

Table 1 . Cost Model Parameters

The parameters calculated by using the above listed cost model parameters are as follows: The number of the total references from class C_i to class C_{i+1} through attribute A is denoted by totlinks(A,C_i,C_{i+1}) and given by the following equation :

$$\text{totlinks}(A,C_i,C_{i+1}) = \text{fan}(A,C_i,C_{i+1}) * \lvert C_i \rvert$$

The probability that an instance of class C_{i+1} is referenced by the instances of class C_i through attribute A is given by the following formula:

$$hitprb(A,C_i,C_{i+1}) = totref(A,C_i,C_{i+1}) / |C_{i+1}|$$

The shorthand notation for these parameters as follows:

$$hitprb_i(A)= hitprb(A,C_i,C_{i+1})$$
$$totlinks_i(A) = totlinks(A,C_i,C_{i+1}).$$

2.1 Selectivity

Selectivity is a parameter used with a predicate to denote the ratio of the elements of a collection satisfying the predicate. When optimizing the queries, selectivity of a predicate is estimated assuming that the values are uniformly distributed. The traditional uniformity and randomness assumptions about value distributions and object placements tend to overestimate costs. However the more sophisticated techniques require more statistical information about the database. The question of how to maintain such information within tolerable overhead is not yet fully resolved [Jar85].

A simple predicate in the system is a triplet of the form <P, Θ, oprnd>, where P is a path expression, Θ is a comparison operator (=, <>, >=, <=, >, <), and oprnd is either a constant or another path expression.

2.1.1 Selectivity for Atomic Attributes

The well-known selectivity calculations described in [Ozk 90] that assume a uniform distribution of the atomic values will be used throughout the derivations presented in this paper.

i. The selectivity of the predicate <p.A, = ,constant> denoted $f_s(p.A,=)$, where p is a variable bound to a class C_i, and A is an atomic attribute, is given by the following formula.

$$f_s(p.A,=) = 1 / dist_i(A)$$

ii. The selectivity of the predicate < p.A, > ,constant> is

$$f_s(p.A,>) = (max_i(A) - constant) / (max_i(A) - min_i(A))$$

iii. The selectivity of the predicate <p.A, BETWEEN, $constant_1$ and $constant_2$ >, where $constant_1$ is less than $constant_2$, is

$$f_s(p.A, BETWEEN) = (constant_2 - constant_1) / (max_i(A) - min_i(A))$$

The parameterless methods of a class C_i with atomic return types are treated in the same manner as the atomic data members of that class.

2.1.2. Selectivity of Path Expressions

Assume that there is a path expression that contains m attributes, A_1 through A_m, where A_1 through A_{m-1} is constructed using the set and the reference constructors. A_m is an atomic attribute and A_i is an attribute of class C_i. We define the selectivity, $f_s(p.A_1.A_2...A_m,\Theta)$, for the path expression predicate "$p.A_1.A_2...A_m\ \Theta\ c$", where Θ is a comparison operator and c is a constant, as the ratio of the elements of the starting collection (to which p is bounded) satisfying the predicate.

The calculation of the selectivity of the predicate $<p.A_m ,\Theta, c>$, $f_s(p.A_m, \Theta)$, is clear from the previous section. Therefore the expected number of instances of C_m, denoted by k_m, that satisfies this condition is:

$$k_m = |C_m| * f_s(p.A_m, \Theta)$$

Consider the following problem: When we start with k objects in class C_1 and follow $k*fan_1(A_1)$ links through attribute A_1 into class C_2, how many objects do we obtain in class C_2? Note that the total number of objects referenced in class C_2 through attribute A_1 of class C_1 is $totref_1(A_1)$. This question can be reduced to the following statistical problem: Given n objects uniformly distributed over m colors, how many different colors c are expected to be selected if we take r objects? This statistical problem has been solved by using different mathematical approximations. An approximation assumed in [Cer 85] is as follows:

$$c(n,m,r) = \begin{cases} r & , r < m/2 \\ (r+m)/3 & , m/2 \leq r \leq 2m \\ m & , r > 2m \end{cases}$$

When we generalize this assuming that we start with k objects of class C_1 and traverse the path $p.A_1.A_2...A_i$ in forward direction, the expected number of objects of class C_{i+1} , denoted by fref, is given by the following formula:

$$fref(p.A_1.A_2...A_i,k) = \begin{cases} k & , i = 0 \\ c(totlinks_i(A_i), totref_i(A_i), fref(p.A_1.A_2...A_{i-1} ,k) * fan_i(A_i)), & i > 0 \end{cases}$$

Note that [Car 75] and [Yao 77], in approximating the number of block accesses to a file for a given query, has defined better approximations to this statistical problem. However, we have calculated and compared the average error per unit time, and found out that c(n,m,r) serves our purposes well.

Starting with one instance of class C_1 , the number of objects of class C_m obtained at the end of forward path traversal is given by $fref(p.A_1...A_{m-1},1)$ which will be denoted as set S_1. On the other hand, there are k_m objects satisfying the predicate $<p.A_m,\Theta,c>$ in the extent of C_m and this set will be denoted by S_2 . The number of objects that are referenced by the objects in the class C_{m-1} in S_2 can be calculated as k_m*hitprb(A_{m-1},C_{m-1},C_m). Denoting this set as S_3, Figure 1 illustrates the set containment relationship among S_1, S_2 and S_3 assuming that S_1 intersects with S_2 and S_3 although this is not the case always.

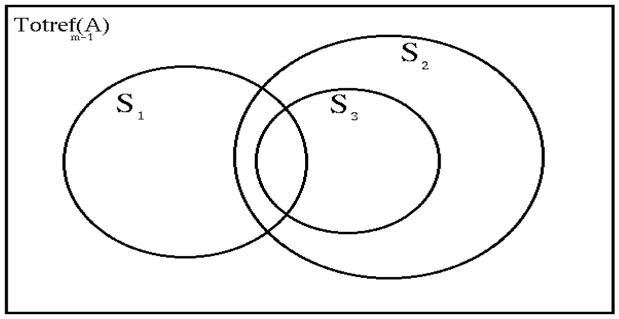

Figure 1 Relation between S_1 , S_2 and S_3

It is clear that if the sets S_1 and S_3 intersects then the object we have started with in C_1 satisfies the predicate <p.A_1...A_m,Θ,c>. Therefore, the selectivity of a path expression can be defined as the probability that the sets S_1 and S_3 will intersect. This probality, $P_{intersect}$, can be calculated as follows:

$$P_{intersect} = o(totref_{m-1}(A_{m-1}), fref(p.A_1.A_2...A_{m-1},1), k_m * hitprb(A_{m-1},C_{m-1},C_m))$$

where o(t,x,y) is the probability that there exists at least one object in common in two sets having cardinalities x and y respectively (x, y ≤ t). Elements of these sets are selected without replacement from the same set of t distinct objects seperately (i.e. after the construction of first set, its elements are placed back to the original set for the selection of the second set). o(t,x,y) is defined as:

$$o(t,x,y) = \begin{cases} 1 & ,x+y>t \\ 1 - C(t-x,y) / C(t,y) & ,\text{otherwise} \end{cases}$$

where C(u,v) stands for the number of combinations of u objects selected from a set of v objects without replacement.

Thus the selectivity of a path expression $f_s(p.A_1.A_2...A_m , \Theta)$ is:

$$f_s(p.A_1.A_2...A_m,\Theta) = P_{intersect}$$

2.1.3. Estimating the Size of a Join

Assume that a join predicate is defined on two path expressions as follows:

$$p.A_1.A_2...A_m = s.B_1.B_2...B_n.$$

A_i is an attribute of C_i, and B_j is an attribute of D_j. The number of instances of C_1 and D_1 are denoted by t_p and t_s, respectively. Notice that if there are no previous selections on these classes, then $t_p = |C_1|$ and $t_s = |D_1|$.

Estimation of the size of a join size necessary for the optimizer to make correct estimates of the costs of alternative query execution plans. The join size will be estimated for three cases.

Case 1. Object Identifier Equality

Our cost model considers the class extensions and an object can be an instance of only one class. Therefore when we consider the object identifier equality, it is clear that A_m and B_n reference the same class, say E. For one instance of C_1, the expected number of instances of E obtained by traversing the path expression starting with the bind variable p is

$$k_p = \mathrm{fref}(p.A_1.A_2...A_m,1)$$

In the same way,

$$k_s = \mathrm{fref}(s.B_1.B_2...B_n,1)$$

We will define the selectivity of a join predicate defined on two path expressions:

$p.A_1.A_2...A_m = s.B_1.B_2...B_n$, where both A_m and B_n contain object identifiers.

Starting with one instance of class C_1, the number of objects of class E obtained at the end of forward path traversal is given by k_p. However not all of these objects may be hit by the links coming from class D_n. Thus k_p*hitprb(B_n,D_n,E) gives the size of the set whose elements are also hit by the links coming from class D_n. Similarly k_s*hitprb(A_m,C_m,E) gives the size of the set which is obtained by starting with one instance of class D_1 and following the path $s.B_1.B_2...B_n$ and whose elements are also hit by the links coming from class C_m. Notice that both of these sets are contained in the set whose cardinality is totref(A_m,C_m,E)*totref(B_n,D_n,E)/|E|. Notice also that totref(B_n,D_n,E)/|E| is hitprb(B_n,D_n,E). The probability, P_i, that these two sets intersects is given by the following formula:

$$P_i = o(\mathrm{totref}(A_m,C_m,E)*\mathrm{hitprb}(B_n,D_n,E),\ k_p*\mathrm{hitprb}(B_n,D_n,E\ ),\ k_s*\mathrm{hitprb}(A_m,C_m,E)).$$

We define the selectivity, q, of a join predicate defined on two path expressions $p.A_1.A_2...A_m = s.B_1.B_2...B_n$ as the probability of an object being in the intersection of two sets with cardinalities

k_p*hitprb(B_n,D_n,E) and k_s*hitprb(A_m,C_m,E) selected from a set whose cardinality is totref(A_m,C_m,E)*hitprb(B_n,D_n,E) with replacement.

$$q=P_i * (k_p*hitprb(B_n,D_n,E)*k_s*hitprb(A_m,C_m,E)/(totref(A_m,C_m,E)*hitprb(B_n,D_n,E))**2).$$

The size of the resulting join, hence, is calculated to be:

$$join\_size = fref(p.A_1.A_2...A_m,t_p) * fref(s.B_1.B_2...B_n,t_s) * q.$$

Case 2. Value-based Equality

After forward traversing the path $p.A_1.A_2...A_{m-1}$ starting with one instance of class C_1, we have obtained k_p objects in class C_m. We need to find out the number of non-null distinct values of attribute A_m corresponding to k_p objects. There is a total of $|C_m|$* notnull(A_m, C_m) number of non-null values of A_m distributed over dist(A_m,C_m) number of distinct values. When we select k_p* notnull(A_m, C_m) number of values out of $|C_m|$* notnull(A_m, C_m) values, the number of non-null distinct values of attribute A_m corresponding to k_p objects, denoted by k_p' is:

$$k_p' = c(|C_m|* notnull(A_m, C_m), dist(A_m,C_m), k_p* notnull(A_m, C_m)).$$

and similarly,

$$k_s' = c(|D_n|*notnull(B_n, D_n), dist(B_n,D_n), k_s*notnull(B_n, D_n)).$$

Then the range of the two-path value-based join is defined to be

$$range_{join} = [max(min(A_m , C_m), min(B_n , D_n)), min(max (A_m , C_m), max(B_n , D_n))]$$

On the other hand, range of each path expression is as follows:

$$range_p = [min(A_m,C_m), max(A_m,C_m)],$$
$$range_s = [min(B_n,D_n), max(B_n,D_n)].$$

Number of distinct values of k_p objects which fall into the $range_{join}$, denoted by $Dist_p$ is calculated as:

$$Dist_p = k_p' * len(range_{join}) / len(range_p), \quad where \quad len(range) = i-j, \quad and \quad range=[j,i].$$

Similarly,

$Dist_s = k_s' * len(range_{join}) / len(range_s)$ gives the number of distinct values of k_s objects falling into the $range_{join}$. The probability, P_i, that these two distinct values intersect is given as follows:

$P_i = o(\text{len}(\text{range}_{join}), \text{Dist}_p, \text{Dist}_s)$

The selectivity of the value-based two-path join, q', is defined as the probality of the value of an object being in the intersection of two sets with cardinalities k_p'*len(range_{join})/len(range_p) and k_s'*len(range_{join})/len(range_s) selected from a set whose cardinality is len(range_{join}) with replacement and it is calculated as:

$q' = P_i * (k_p' * \text{len}(\text{range}_{join})/\text{len}(\text{range}_p)) * (k_s' * \text{len}(\text{range}_{join})/\text{len}(\text{range}_s)) / \text{len}(\text{range}_{join}) ** 2$

Then, estimated join size is

$\text{join\_size} = \text{fref}(p.A_1.A_2...A_m, t_s) * \text{fref}(s.B_1.B_2...B_n, t_p) * q'.$

Case 3. Non-equality Joins

For such joins, estimated join size is taken as the cardinality of the cartesian product of two collections obtained by traversing the links in the path expressions, which is the result of a pessimistic assumption.

$\text{join size} = \text{fref}(p.A_1.A_2...A_m, t_s) * \text{fref}(s.B_1.B_2...B_n, t_p)$

3. Cost Analysis of Operators in a Paging Environment

In this section a number of parameters will be defined which are used in the analysis of the costs of forward and backward traversals. These parameters are the costs of sequential, random and indexed accesses.

In Table 2 the physical parameters of a disk which are used in the numerical examples are provided.

Parameter	Definition
B	Block size
btt	Block transfer time
ebt	Effective block transfer time
r	Avarege rotational latency
s	Avarage seek time

Table 2. Physical Parameters for hard disk

The cost of sequential accesses to *b* pages is denoted by *SEQCOST(b)* and is calculated as

$$SEQCOST(b) = s + r + b * ebt .$$

The cost of random access to *b* pages, denoted by *RNDCOST(b)* is

$$RNDCOST(b) = b * (s + r + btt).$$

In Table 3, the information kept by the system for a B^+-tree index I is shown.

Parameter	Definition
v(I)	Order of the B^+ tree
level(I)	Number of levels
leaves(I)	Number of pages at the leaf level
keysize(I)	Size of the key value
unique(I)	Unique flag

Table 3. Parameters for a B^+-tree

The cost of accessing object identifiers of *k* random keys by using a B^+ tree secondary index *I* is calculated as follows:

The number of pages at the leaf level of the B^+ tree is given as leaves(I). Assuming there are 2v keys at each page and also assuming that the leaves are ln2 full [Yao 78], the total number of keys at the leaf level is leaves(I) *2v*ln2. Then number of leaf level pages to be fetched in order to retrieve k keys is given by:

$$c(n_1,m_1,r_1)=c(\text{leaves}(I)*2v*\ln 2, \text{leaves}(I), k)$$

In other words, there are leaves(I)*2v*ln2 keys distributed over leaves(I) pages, and the question is, how many pages should we retrieve in order to get k keys from the leaf level of the B^+ tree index. Generalizing this to the upper levels of the secondary B^+ tree we obtain the formula, *INDCOST(k)*, giving the cost of indexed access in terms of pages to be retrieved.

$$INDCOST(k) = \left(\sum_{i=1}^{level(I)} c(n_i, m_i, r_i) \right) * RNDCOST(1)$$

where n_i = leaves(I) / $(2v*\ln 2)^{i-2}$, m_i = leaves(I) / $(2v*\ln 2)^{i-1}$, and

$$r_i = \begin{cases} k & , \ i = 1 \\ c(n_{i-1}, m_{i-1}, r_{i-1}) & , \ \text{otherwise.} \end{cases}$$

Finally, the cost of a range query using a secondary B^+-tree index *I*, given by *RNGXCOST(fract)*, is given by the equation

$$RNGXCOST(fract) = fract * leaves(I) * (s + r + btt)$$

where *fract* is the proportion of the objects in the given range to the whole domain.

3.1 Cost of Forward Traversal

Assume that there exists a path expression, $p.A_1.A_2...A_m$, where A_1 through A_m are constructed using set, list or reference contructors, A_i is an attribute of C_i and A_m references class C_{m+1}. We will calculate the cost of forward traversal for this expression. We use the short-hand notation for fan_i, $totref_i$ and $totlinks_i$ given previously.

Given k_1 object identifiers of instances of C_1, it is necessary to find out the cost of forward traversing a given path. In this analysis, the following assumptions are made:

i. No pages of an instance of C_i is in the buffer, where i>1.
ii. To eliminate the complexity of taking the effect of the page replacement policy of the buffer management system into account, the available buffer space is assumed to be large enough to hold all of the pages fetched during the traversal.

Let k_i be the number of distinct instances of C_i which are referenced by attribute A_{i-1} of C_{i-1}. Then, after following the links from C_1 to C_2 for k_1 objects in C_1, the number of objects in C_2 , is given by k_2 :

$$k_2 = c(totlinks_1(A_1), totref_1(A_1), k_1 * fan_1(A))$$

If there is a binary join index between the classes C_1 and C_2, which is implemented as a pair of B+ trees, through attribute A_1 then the cost of forward traversing into these k_2 items is just the cost of dereferencing them since the contents of k_1 objects from C_1 will not be accessed. Then, the cost function, bjicost, is written as:

$$bjicost = INDCOST(k_1)$$

If there are no binary join indices, on the other hand, the cost is the total cost of accessing the pages over which k_1 objects of C_1 are distributed, together with the cost of accessing the blocks in which set information on object identifiers are stored, if A_1 is a set. In this case, the cost is given as:

$$rcost = RNDCOST(nbpg) + k_1 * SETINFO(I)$$

where

$$nbpg = nbpages(C_1) * (1 - (1 - 1 / nbpages(C_1))^{k1}) \quad .$$

and nbpg gives the number of pages to be fetched from class C_1 to retrieve k_1 objects, where $|C_1|$ objects are distributed over *nbpages*(C_1)[Car 75].

If the type constructor of attribute A_1 is of REFERENCE type, then obviously the second term of the equation for rcost, i.e. $k_1 * SETINFO(I)$, disappears. The parameter *SETINFO(I)* is a value proportional to the average cost of accessing the elements of a set organized as a B^+-tree and it is as given below:

$$SETINFO(I)=(level(I)+leaves(I))*RNDCOST(1)$$

The minimum of *bjicost* and *rcost* is chosen and added to the total forward traversal cost. Then the procedure for calculating the cost of a forward traversal of a path expression is as given below:

```
function ftracost( int $k_1$, PathExpression p.$A_1$.$A_2$...$A_m$ )
{
   /* $k_1$ is the initial number of object identifers of instances of class $C_1$ */

function ftracost
    int i = 1;
    float fcost=0, bjicost, rcost;

    while( i ≤ m) {
       If (there are forward binary join indices on $A_i$ )  bjicost = INDCOST($k_i$);
       else bjicost = ∞ ;
       nbpg = nbpages($C_i$) * ( 1 - ( 1 - 1/nbpages($C_i$) )$^{ki}$ );
       switch( the constructor of $A_i$ ) {
             case SET :
                rcost = RNDCOST(nbpg) + $k_i$ * ( SETINFO(I) );
             case REF  :
                rcost = RNDCOST(nbpg);
       } /* switch */

       fcost = fcost + min(bjicost,rcost);
       i = i + 1;
       $k_i$ = c(totlinks$_{i-1}$, totref$_{i-1}$, $k_{i-1}$ * fan$_{i-1}$);
    }   /* while  */

    return(fcost);

}   /* function  */
```

3. 2 Cost of Backward Traversal

In our object model, there are no backward links. If there is binary join index between C_i and C_{i+1}, the costs are calculated as in the forward traversal. However, if there are no binary join indices, instances in C_i are selected by sequential scan of C_i. The number of instances, k_i, selected in C_i as the result of this operation is estimated as:

$k_i = |C_i| * \text{notnull}(A_i, C_i) * o(\text{totref}_i, \text{fan}_i, k_{i+1})$, where k_{i+1} is the number of instances in C_{i+1} selected in the backward traversal of the path expression and $o(\text{totref}_i, \text{fan}_i, k_{i+1})$ is the probability is that an object in C_i gives reference to an instance in this set.

Given a path $p.A_1.A_2...A_m$, and keeping the assumptions made in the previous section, an analoguous algorithm for calculating the cost of backward traversal is given below.

```
function btracost( int $k_m$, PathExpression $p.A_1.A_2...A_m$ )
/*  $k_m$ is the initial number of object identifers of instances of class $C_m$ */
int i;
float bcost=0, bjicost, scost;
i=m;
while ($i \neq 0$) {
  $k_i = k_i$ * hitprb($A_{i-1}$,$C_{i-1}$,$C_i$);
  bjicost=$\infty$;
  If (there are binary join indices on $A_i$ )
     bjicost = INDCOST($k_i$);
  scost = SEQCOST(nbpages($C_i$)) ;
  bcost = bcost + min(bjicost,scost);
  i = i - 1;
  $k_i = |C_i|$ * notnull($A_i$, $C_i$) * o($\text{totref}_i$, $\text{fan}_i$, $k_{i+1}$) ;
 }    /* while */
  return(bcost);
}    /* function */
```

4. An Example

In order to provide a better understanding to the use of the cost formulas introduced in the previous sections, the following example is provided.

Assume the following SQL query:

```
SELECT  c.name
FROM Company c
WHERE c.division.staff.drives.color=’blue’
          and c.location.country= ’USA’
```

The physical parameters values of a typical disk drive is given in the table 4.

Parameter	Value
B	2 KByte
btt	0.8 msec
ebt	0.84 msec
r	8 msec
s	16 msec

Table 4. Physical Parameters values for a typical hard disk

Let us also assume the database statistics are as shown in Table 5, Table 6 and Table 7.

Class	|C|	nbpages(C)	size(C)
Company	30	30	2 KByte
Division	50	50	2 KByte
Employee	15000	45000	6 KByte
Vehicle	60000	150000	6 KByte
City	30	30	2 KByte

Table 5 . Statistics on the example database

Class	Attribute	dist
Vehicle	color	10
City	country	10

Table 6. Statistics on the example database

Class	Attribute	fan	totref	totlinks	hitprb
Company	division	4	40	120	0.80
Division	staff	150	7000	7500	0.47
Employee	drives	4	50000	60000	0.83
Company	location	1	30	30	1.00

Table 7. Statistics on the example database

The sequential scan costs of the class extensions are as given in Table 8.

Class	Sequential Scan Cost(secs.)
Company	0.0495
Division	0.0663
Employee	37.8243
Vehicle	126.0243
City	0.0495

Table 8. Sequential scan costs of the class extensions

The selectivities of the path expressions are calculated by the following formulas:

f_s(P1)= f_s(c.division.staff.drives.color, =) = 1.00

f_s(P2)= f_s(c.location.country, =) = 0.1

By using the cost formula for backward traversal, btracost, the costs of the backward traversals of the path expressions P1and P2 are calculated to be:

b_{P1} = Sequential scan cost of Vehicle to evaluate "color= ’blue’ "+
backward traversal of c.division.staff.drives

= 126.0243 + btracost(f_s(color,=)*|Vehicle|, c.division.staff.drives)
= 126.0243 + 37.9401
= 163.9644 secs

b_{P2} = Sequential scan cost of City to evaluate "country= 'USA' "+
backward traversal of c.location
= 0.0495 + btracost(f_s(country,=)*|City|, c.location)
= 0.0495 + 0.0495
= 0.099 secs

It should be noted that there can be some fast access paths defined on the classes like access support relations [Kem 90] which are redundant seperate structures to store object references that are frequently traversed or path indices [Kim 90, Ber 93] which store instantiations of a path, that is, sequences of objects. We will assume that there are no such indices on the classes. Then, some of the available alternatives to be checked by the query optimizer are as follows:

1. A backward traversal of the predicate P1 followed by a forward traversal of the path expression P2.

The cost of this alternative is computed to be:

c1 = b_{P1} + ftracost(f_s(P1) * |Company|, c.location)
= 163.9644 + 0.9628
= 164.9272 secs

2. A backward traversal of the path expression P2 followed by a forward traversal of the path expression P1. The cost of the second alternative is expressed as follows:

c2 = b_{P2} + ftracost(f_s(P2) * |Company|, c.division.staff.drives)
= 0.099 + 119.024
= 119.123 secs.

3. Converting all implicit joins in the path expressions to the explicit joins, and executing these joins by using a pointer-based join algortihms like hybrid hash partitioning join or sort-merge join[She 90]. No matter what join technique is used, it will require at least one sequential scan of the file. Therefore the minimum cost for this alternative is greater than the sum of sequential scans of the extensions of the classes.

c3 >= 163.9644 secs.

It is clear from this example that for the optimizer to decide on the best execution plan among the available alternatives, it is in need of the selectivities and the costs of path expressions.

4. Conclusions

The query optimizer needs to know the cost of possible alternatives to choose the best one. The selectivities of attributes and the selectivities and the cost of explicit joins for relational data model are well-known. These formulas are also used for object-oriented query processing. However path traversals, that is, executing precomputed joins is one of the possible alternatives in executing object-oriented queries. In this paper we have provided the formulas for calculating the selectivities of path expressions, for estimating size of an explicit join involving path expressions, and also for calculating the cost of forward and backward traversals.

References

[Alh 93] Alhajj, R, Arkun, M. E., "A Query Model for Object-Oriented Databases", Proc. IEEE Conf. on Data Eng., 1993.

[Atk 92] Atkinson, M., Bancilhon, F., DeWitt, D., Dittrich, D., Maier, D., Zdonik, S., "The Object-Oriented Database System Manifesto", in Building an Object-Oriented Database System (F. Bancilhon, C. Delobel, and P. Kanellakis, Edtrs.), Morgan-Kaufmann, 1992.

[Ber 93] Bertino, E., Martino, L., Object-Oriented Database Systems: Concepts and Architectures, Addison-Wesley, 1993.

[Ber 92] Bertino, E., Foscoli, P., "A Model of Cost Functions for Object-Oriented Queries", In Proc. of 5th International Workshop on Persistent Object Systems, Italy, September 1992.

[Bla 93] Blakeley, J. A., McKenna, W. J., Graefe, G., "Experiences Building the Open OODB Query Optimizer", Proc. of the ACM SIGMOD Conf., 1993.

[Car 86] Carey, M., DeWitt, D.. Richardson, J., Shekita, E., "Object and File Management in EXODUS Extensible Database System", in Proc. of the 12th Intl. Conf. on VLDB, 1986.

[Car 88] Carey M.J., DeWitt D.J., Vandenberg S.L., "A Data Model and Query Language for EXODUS", Proc. of the ACM SIGMOD Conf., 1988.

[Car 75] Cardenas A.F., "Analysis and Performance of Inverted Data Base Structures", Comm. ACM, May 1975.

[Cer 85] Ceri, S., Pelagatti, G., Disributed Database systems, McGraw Hill, 1985

[Clu 92] Cluet S., Delobel C., "A General Framework for the Optimization of Object-Oriented Queries", Proc. of the ACM SIGMOD Conf. on Management of Data, 1992.

[Dar 92] Dar S., Gehani N.H., Jagadish H.V., "CQL++: A SQL for the Ode Object-Oriented DBMS",in Proc. of Extending Database Technology, 1992.

[Deu 91] Deux, O., et al., "The O2 System", Comm. of the ACM, Vol. 34, No.10, 1991.

[Jar 84] Jarke M., Koch J., "Query Optimization in Database Systems", Computing Surveys, Vol. 16, No. 2, 1984.

[Jar 85] Jarke M., Koch J., Schmidt J. W., "Introduction to Query Processing", Query Processing in Database Systems, Springer-Verlag, 1985, pp 3-28.

[Jen 90] Jeng, B. P., Woelk, D., Kim, W., Lee, W-L., "Query Processing in Distributed ORION", In Proc. of Extending Data Base Technology, 1990.

[Kel 91] Keller, T., Graefe, G., Maier, D.," Efficient Assembly of Complex Objects", Proc. of the ACM SIGMOD

Conf., 1991.

[Kem 90] Kemper A., Moerkotte G., " Access Support in Object Bases", Proc. of the ACM SIGMOD Conf., 1990.

[Kim 90] Kim W., Introduction to Object-Oriented Databases, The MIT Press, 1990.

[Lan 91] Lanzelotte, R. S. G., Valduriez, P., Ziane, M., Cheiney, J-P., "Optimization of Nonrecursive Queries in OODBs", In Proc. of the Second Intl. Conf. on Deductive and Object-Oriented Databases, 1991.

[Ozk 90] Ozkarahan E., "Database Management Concepts, Design and Practice", Prentice-Hall, 1990.

[Ozk 92] Ozkan C., Evrendilek C., Dogac A., " Optimization and Implementation of MOODSQL", Software Research and Development Center, TUBITAK, Technical Report No.8, September 1992.

[Sal 88] Salzberg B., "File Structures, an Analytic Approach", Prentice-Hall International, Inc, 1988.

[Sha 90] Shaw, G. M., Zdonik, S. B., "A Query Algebra for Object-Oriented Databases", Proc. IEEE Conf. on Data Eng., 1990.

[She 90] Shekita, E. J., Carey, M. J., "A Performance Evaluation of Pointer-Based Joins", Proc. of the ACM SIGMOD Conf., 1990.

[Str 90] Straube, D. D., Ozsu, M. T., "Queries and Query Processing in Object-Oriented Database Systems", ACM Trans. on Inf. Sys. 8, 4, 1990.

[Ull 88] Ullman J. D., Principles of Database and Knowledge-Base Systems, Vol 2, Computer Science Press, 1988.

[Yao 77] Yao S.B., "Approximating Block Access in Database Organizations", Comm. of the ACM, Vol. 20, No. 4, April 1977.

[Yao 78] Yao, A., "Random 3-2 Trees", Acta Informatica, Vol.9, No.2, 1978.