Semantic Cardinality Estimation for Queries over Objects †

Marian H. Nodine, Mitch Cherniack, Mark H. Nodine
{mhn,mfc}@cs.brown.edu, nodine@mcrc.mot.com
Keywords: Query optimization, cardinality estimation, cost models, query algebras

Abstract

In this paper, we address the problem of estimating cardinalities of queries over sets of objects. We base our estimates on a knowledge of subset relationships between sets (e.g., `Students` form a subset of `People`). Previous work on cardinality estimation has assumed that every subset is a representative sample of the set it is taken from. We maintain (offline) information for the cases where the subset is not a uniform representative, and use this information to improve the accuracy of our cardinality estimates.

We present cardinality estimation techniques for *image sets* and *select sets*. Image sets are created by applying a function to an existing set. Select sets contain the elements that satisfy some predicate in an existing set. Empirically we show that we can obtain estimates within a factor of four for reasonable image sets on our test database while maintaining offline statistics requiring space linear in the number of named sets in the database and the number of attributes defined per type. We also present an approach for estimating cardinalities of select sets. Our approach is orthogonal to statistical sampling approaches that give a more accurate estimation of the distribution of values an attribute can take in a set, but do not deal with semantically meaningful subsets whose attributes may not have the same distribution.

1 Introduction

The development of advanced databases such as nested relational databases, complex object bases and object-oriented databases has been motivated by the limited modeling capabilities of other database paradigms. Expressive modeling power is partially achieved by extensible data types and type relationships such as subtyping. Unfortunately, this expressiveness directly impacts the use of heuristics in query optimization. In this paper, we show how information about meaningful data subsets (such as those which are the extents of subtypes) can be used to guide the optimization of an object-oriented query.

Persistent object systems by their nature maintain information about sets and their subsets. For example, given a type hierarchy we know that the extent of some type `T` is a superset of the extents of each subtype of `T`. Subtype extents are a special case of *semantically meaningful subsets*. A semantically meaningful subset is **not** a representative sample of its supersets but differs by some characteristic in a statistically significant way. For example, the set of `Students` by nature is distinct from the general set of `People` with respect to age. Other semantically meaningful subsets are formed as a result of queries.

In an advanced database, sometimes the only way to decide whether a transformation would improve the execution time of a query is to compare estimated costs for the original and transformed queries. In this paper, we examine cardinality estimation, which is a prerequisite to accurate cost estimation. We describe

†Partial support for this work was provided by the Advanced Research Projects Agency under contract N00014-91-J-4052 ARPA order 8225, and contract DAAB-07-91-C-Q518 under subcontract F41100.

techniques to use set and subset information to improve the accuracy of cardinality estimation. We keep information on sets and subsets in an *extent table*. This information includes, for each known extent (and possibly other subsets), its cardinality and its smallest known superset. Also, for each attribute, the extent table includes information used to estimate the number of distinct values that attribute takes in the set.

The extent table helps by allowing us to track natural, non-representative subsets. We can estimate the cardinality of a query result by determining the smallest set that contains the result (the result's *smallest known superset*, or SKS) and using its cardinality information to extrapolate the result set cardinality. If the result's SKS is not much bigger than the result, this should lead to very accurate cardinality estimates.

Also, this approach allows us to maintain better domain information for an attribute. One problem in estimating selectivities is determining the logical domain over which users are going to choose the values to select against. For instance, when determining the selectivity of all the `Students` that have name n, the two potential domains for n are (1) the set of student names and (2) the set of character strings. Neither is a satisfactory approximation. In our scheme we approximate the logical domain of n as the domain of the "name" attribute of `Students`' most general supertype, `People`.

This paper is organized as follows. We present our data model and assumptions in Section 2. We describe our algorithms for maintaining extent information in Section 3. In Section 4, we present our cardinality estimation techniques for mapping[1] and selection queries. Section 5 shows experimental results validating our estimation techniques for mapping query size estimation. We discuss related work in Section 6 and conclude in Section 7.

2 Data Model, Definitions, and Basic Assumptions

2.1 Data Model

We assume an object-like model for the data in our database. This model incorporates features of the nested relational and complex object data models.[2] Every entity in our model is an object. Objects in our model are one of the following:

- *Base objects* are `Boolean`, `String` or `Integer` values.[3]
- *Complex objects* belong to user-defined complex types. These types name sets of *attribute selectors* which when invoked return associated objects.
- *Pair objects* `[`e`,` e'`]` where e and e' are base, complex or pair objects.
- *Function and predicate objects* are invocable with single arguments. We denote arbitrary functions (predicates) with variable names f and g (p and q). Specific functions and predicates are referred to by their bodies, italicizing implicitly lambda-bound variables.

[1] Here, mapping applies a function to the elements of a set, and subsumes relational projection.
[2] In fact, this model incorporates object-oriented features with the exception that we do not deal with object mutability.
[3] Currently, we do not support `Real` values in base objects because a bounded set of reals still has an infinite number of values, so our methods for maintaining type extents for `Integers` fail with `Reals`.

Function / Predicate	Example Denotation	Meaning
identity	x	$\lambda\ (x)\ x$
constant	c (c a constant)	$\lambda\ (x)\ c$
path expression	$x.\vec{\mathrm{f}}$	$\lambda\ (x)\ x.\vec{\mathrm{f}}$
path expression = value	$x.\vec{\mathrm{f}} = c$ (c a constant)	$\lambda\ (x)\ x.\vec{\mathrm{f}}\ =\ c$
path expression $\in$ Set	$x.\vec{\mathrm{f}} \in S$ (S a set)	$\lambda(x, y)\ x.\vec{\mathrm{f}}\ \in\ S$
path expression$_1$ = path expression$_2$	$x.\vec{\mathrm{f}} = x.\vec{\mathrm{g}}$	$\lambda(x, y)\ x.\vec{\mathrm{f}} = x.\vec{\mathrm{g}}$
path expression$_1$ $\in$ path expression$_2$	$x.\vec{\mathrm{f}} \in x.\vec{\mathrm{g}}$	$\lambda(x, y)\ x.\vec{\mathrm{f}} \in x.\vec{\mathrm{g}}$

Table 1: Allowable bodies of function and predicate arguments to mapping and selection queries.

- *Sets* are homogeneous collections of base, complex, or pair objects. We denote sets using capital letters A and B. We use the standard notation $|A|$ to denote the cardinality of a set. We use lower-case letters and primes a, a' to denote elements of the set.

In this paper, we use a sample schema with the types `Person`, `Address` and `City`. `Person` has attributes `firstname, lastname, address, gender, age, spouse` and `numRoommates`. `Address` has attributes `number, street` and `city`. `City` has attributes `name, state, zip` and `mayor`. We assume that attribute selectors are *unary*, and invoked using dot notation (as in `p.address`). Attribute selectors can be composed to form *path expressions*, as in `p.address.street`. Vector notation (as in $\mathtt{x}.\vec{\mathtt{f}}$) denotes an arbitrary path expression.

Our model accounts for two set constructions:

- *Type Extents* are assumed for all base and complex object types. Type extents for base types (named `Integers, Booleans and Strings`) include constants used in the database. Type extents for complex types include all constructed objects of that type. For a given type extent, if the type's parent is `Object`, it is a *root type extent*. If the type has as its ancestor some root type, the extent is a *subtype extent*. Subtype extents include references to the smallest known superset.

- *Query Sets* in this paper are restricted to *image sets* and *select sets*.[4] An image set $\tilde{\pi}_f(A)$ is formed by applying the unary function f to each element in the set A, giving the set $\{f(a)|a \in A\}$. If f is a path expression, we also refer to the set $\tilde{\pi}_f(A)$ with the notation $A.\vec{\mathrm{f}}$. A select set $\tilde{\sigma}_p(A)$ is formed by filtering out the elements that satisfy the predicate p from the set A, giving the set $\{a|a \in A \wedge p(a)\}$ Table 1 lists the forms of functions and predicates allowed in queries.

2.2 Definitions: Selectivity and Projectivity

The *projectivity* of a function over the elements of a set is the ratio of the size of its range to the size of its domain. Formally:

[4] Other query sets have been addressed using this approach, and are written up in [NCN95].

Definition 2.1 (Projectivity) *For any function f and non-empty set A, proj $(f, A) = \frac{|\tilde{\pi}_f(A)|}{|A|}$.*

The *selectivity* of a predicate on a set is the ratio of the number of selected elements of the set to the size of the set. Formally:

Definition 2.2 (Selectivity) *For any predicate p over elements of nonempty set A, sel $(p, A) = \frac{|\tilde{\sigma}_p(A)|}{|A|}$.*

2.3 Basic Assumptions: Uniform Ratio and Uniform Distribution

Barring other information, we make a few assumptions about the data in the database. The first is the *uniform ratio assumption*, which is used in cases where we have no information contradicting the assumption that B is a representative subset of A.

Assumption 1 (Uniform Ratio Assumption) *For a given subset B of set A, where A and B are of the same type with attribute f, proj $(f, A) \approx$ proj (f, B)*

The second assumption is the *uniform distribution assumption*:

Assumption 2 (Uniform Distribution Assumption) *For a given set A and path expression $\vec{\mathbf{f}}$, and any two values c_1 and c_2 from the image set $\tilde{\pi}_f(A)$, sel $(a.\vec{\mathbf{f}} = c_1, A) \approx$ sel $(a.\vec{\mathbf{f}} = c_2, A)$*

The uniform ratio assumption is used when alternative distribution information is not available. It also allows us to transform a selectivity expression into a second one that is approximately equal, but is selecting from an image set. For example, by the uniform distribution assumption, we know for a given constant c in the image set $A.\vec{\mathbf{f}}$ that $|\{a \in A \mid a.\vec{\mathbf{f}} = c\}| \cdot |A.\vec{\mathbf{f}}| \approx |A|$. Because $A.\vec{\mathbf{f}}$ is a set, we also know that $|\{a' \in A.\vec{\mathbf{f}} \mid a' = c\}| = 1$. Therefore, we can compute the ratio

$$\frac{|\{a \in A \mid a.\vec{\mathbf{f}} = c\}|}{|A|} \approx \frac{|\{a' \in A.\vec{\mathbf{f}} \mid a' = c\}|}{|A.\vec{\mathbf{f}}|},$$

i.e., *sel* $(a' = c, A.\vec{\mathbf{f}}) =$ *sel* $(a.\vec{\mathbf{f}} = c, A)$. Note that for $c \notin A.\vec{\mathbf{f}}$ this last equality also holds trivially, because both selection operations return the empty set. We use this type of transformation in our deductions of selectivities in Section 4.

3 Extent Tables

In this section, we consider the statistical and semantic information that we maintain about selected sets in the database for use in estimating cardinalities. This information is maintained in an *extent table*. Our concern is to keep enough information in the extent table to allow for reasonable estimates without making it unwieldy to maintain. To this end, we consider what sets to keep in the table, what information to keep for each set, and how to infer statistical and semantic information for sets not in the table.

The crucial information kept in the extent table concerns the sizes and smallest known supersets of sets and the *projectivity* of functions. Every type extent represented in the extent table has an associated *size*

Index	*Name*	*Size*	*Projectivity*	*SKS*	*Comments*
0	Objects	14477	-	0	The set of all objects
1	Strings	1501	-	0	The set of string literals
2	Ints	1001	-	0	The set of all integer constants used
3	Addresses	3500	-	0	Type extent of `Address`
4	Cities	52	-	0	Type extent of `City`
5	People	9959	-	0	Type extent of `Person`
6	Students	4009	-	5	Type extent of `Student`
7	Grads	1056	-	6	Named Set
8	UGrads	2953	-	6	Named Set
9	Faculty	503	-	5	Named Set
	...				
11	Addresses.city	-	0.015	4	Image set
13	Cities.state	-	0.288	1	Image set
20	People.address	-	0.351	3	Image set
25	Students.address	-	0.578	20	Image set
	...				
35	UGrads.address	-	0.505	25	Image set
	...				

Table 2: An Example Level$_1$ Extent Table

field, while image sets ($\tilde{\pi}_f(A)$) have associated projectivity fields ($proj\ (f, A)$). We maintain projectivities for image sets because they are likely to remain fairly stable with respect to set insertions and deletions, while image set sizes may need to be updated more frequently.

3.1 The Sets Kept in the Table

Extent tables can vary in complexity according to how many and which sets are represented. Every extent table at a minimum has entries for all base, root, and subtype extents as well as all image sets $A.\vec{\mathbf{f}}$ where A is a root type extent and $\vec{\mathbf{f}}$ is a path expression of length 1. A *minimal extent table* only has this information, and is the smallest one that can be used to estimate cardinalities for all image and select sets. Additional image sets may be kept in the table. We classify the complexities of non-minimal extent tables as follows:

- A *Full Level$_i$ Extent Table* (for $i \geq 1$) includes entries for all image sets, $A.\vec{\mathbf{f}}$, where $\vec{\mathbf{f}}$ is of length $n \leq i$ and A is a type extent. Table 2 shows a portion of a full level$_1$ extent table based on our example schema. Note that a minimal table would only include the sets indexed from 0–20.

- An *Adaptive Level$_i$ Extent Table* subsumes the minimal extent table and also includes selected image sets for paths of length $n \leq i$. Adaptive level$_i$ extent tables keep track of the subset of the information in the full level$_i$ extent table that cannot be derived accurately from superset information, i.e., minimal extent information plus information on non-representative subsets.

Algorithm 3.1 (Extent Table Maintenance)

```
algorithm update_table (A.f:name of an image set, n:actual size of image set)
    if A.f is not in the extent table then
        use techniques of Section 4.1 to estimate its size, m
        if the estimate is not reasonable,
            add entry for A.f to extent table with projectivity of n/|A| (estimated |A|).
    else
        pretend that A.f is not in the table and use techniques of Section 4.1 to
            estimate its size, m
        if the estimate is reasonable, remove entry for A.f from table
        else update entry for A.f so that projectivity is n/|A| (estimated |A|).
```

Figure 1: Extent Table Maintenance Algorithm

3.2 Adaptive Extent Table Maintenance

We propose a simple algorithm to maintain relevant image set information in an adaptive extent table.[5] We envision the algorithm being executed every time normal database activity such as querying or indexing reveals the actual cardinality of an image set. Thus, the maintenance of the extent table should not generate additional database accesses.

The high-level idea behind the algorithm is simple; we want to add all non-representative sample sets to the extent table as we recognize them, and similarly remove all representative sample sets from the extent table. We remove representative sets to save space, because their cardinalities can be derived easily from other information in the extent table.

The pseudocode for an extent table maintenance algorithm is given in Figure 3.1. Here, we assume the designer can specify his own algorithm for deciding whether or not an estimate is a "reasonable approximation".[6] In Section 5 we show empirically that the adaptive maintenance technique can be used to compute reasonable estimates for all sets without demanding an unreasonably large extent table.

3.3 Inference of Supersets

Even with reasonably complex and complete extent tables, cardinalities and projectivities will need to be estimated in some cases. The algorithm for estimating projectivities and cardinalities is described in Section 4.1. In this section we present algorithms for finding the name of the smallest known superset (SKS) for any image set (Figure 2) and the name of the smallest known common superset (SKCS) of any two image sets (Figure 3). Note that both algorithms take set names as inputs and return set names as outputs. This works because the naming scheme for image sets and extents facilitates the search for the SKS and SKCS. Note also that we use the *index* column of the extent table in standard array-like fashion. Therefore, `Table[5]` refers to the entry describing the extent set `People` while `Table[20].ratio` refers to the ratio,

[5] Cardinalities of type extents might be maintained rigorously by updating the extent table at times of insertions, removals, object creations, removals and assignments of constant values. Off-line sampling could used to estimate cardinalities.

[6] Reasonableness means that its error falls within some bounds specified by some input standard error metric.

Algorithm 3.2 (SKS)

```
algorithm SKS (s:  string) :  string // (s is of the form S.m_1. ... .m_n)
    if for some i, Table[i].name = s return (Table[Table[i].sks].name)
    else return (SKS (S.m_1. ... .m_{n-1})||'.'||m_n)
```

Figure 2: Unary Smallest Known Superset (SKS) Algorithm.

Algorithm 3.3 (SKCS)

```
algorithm SUBSET? (s_1, s_2:  string) :  Bool
    -- return true if s_1 names a set which is a subset of s_2
        if (s_1 = s_2) return true
        else if (s_1 = ''Objects'') return false
        else return SUBSET? (SKS (s_1), s_2)

algorithm SKCS (s_1, s_2:  string) :  string
    -- finds the smallest known common superset of s_1 and s_2
        if SUBSET? (s_1, s_2) then return s_1
        else if SUBSET? (s_2, s_1) then return s_2
        else return SKCS (SKS (s_1), SKS (s_2))
```

Figure 3: Smallest Known Common Superset (SKCS) Algorithm for Pairs of Sets

proj (`address`, `People`). The last line uses string concatenation ($||$) to form the result.

The correctness of the SKS algorithm (Figure 2) comes from the observation that the SKS of any image set, $A.f_1.\cdots.f_n$ is the image set formed by applying attribute selector f_n to the set that is the SKS of $A.f_1.\cdots.f_{n-1}$. For example, the SKS of `Students.address.city` is `People.address.city` which in turn has an SKS which is `Addresses.city`. For all type extents or image sets formed from path expressions in the table, this algorithm will find the SKS directly.

The SKCS algorithm is shown in Figure 3, along with the SUBSET? algorithm, which takes the names of two sets as input and returns true if the first names a set which is a subset of the second. The SKCS algorithm computes the smallest known common superset of its input sets. The correctness of this algorithm follows from both the correctness of the SKS algorithm and the partial ordering of the subset relation on sets with `Objects` as the superset of all sets. According to this algorithm

$$\begin{aligned} \texttt{SKCS(UGrads.address, Faculty.address)} &= \texttt{SKCS(Students.address, People.address)} \\ &= \texttt{People.address.} \end{aligned}$$

4 Estimating Cardinalities

4.1 Image Sets and Projectivities

Our estimation technique estimates the cardinality of an image set $\tilde{\pi}_f(A)$ by first estimating the projectivity *proj*(f, A). Projectivity is either looked up directly in the extent table (for image sets of non-representative

Algorithm 4.1 (Projectivity)

```
algorithm est_proj (f_1. ... .f_n:  String, A: String)
    (f_1. ... .f_n is a path expression, A is the name of a set)
        if A.f_1. ... .f_n is in the extent table then
            return the projectivity entry in the extent table
        else if ∃k < n such that A.f_1.....f_k is in the extent table then
            choose the largest such k and
            return (est_proj (f_1.....f_k, A)) · (est_proj (f_{k+1}.....f_n, A.f_1. ... .f_k)).
        else return (est_proj (f_1.....f_n, SKS(A)))
```

Figure 4: Projectivity Estimation Algorithm. The final else clause uses the uniform distribution assumption because no information to the contrary is available.

subsets) or computed using the uniform ratio assumption (for representative subsets). The cardinality of the result of a mapping query that takes the image of function f on set A is $|A| \cdot proj(f, A)$.

In Figure 4 we present an algorithm for estimating the projectivity of any function of the form $x.\mathtt{f}_1.\cdots.\mathtt{f}_n$. The projectivity of the identity function x is 1. For path expressions of length > 1, we look the projectivity up directly if it is in the table, or compute the projectivity from the most accurate information found in the extent table if it is not. Note that where the uniform ratio assumption does not in fact hold, the result will be incorrect, but the extent table will also be updated with correct information during the processing of the affected query.

Algorithm 4.1 is guaranteed to terminate so long as we have at least a minimal extent table. However, the result will be more accurate when more information is available.

The call of `est_proj(address.city,Students)` in the level$_1$ extent table of Table 2 proceeds as follows:

$$
\begin{aligned}
&\mathtt{est\_proj(address.city, Students)} \\
&\quad = \mathtt{est\_proj(address, Students)} \cdot \mathtt{est\_proj(city, Students.address)} \\
&\quad = \mathtt{est\_proj(address, Students)} \cdot \mathtt{est\_proj(city, Persons.address)} \\
&\quad = \mathtt{est\_proj(address, Students)} \cdot \mathtt{est\_proj(city, Addresses)} \\
&\quad = 0.00867
\end{aligned}
$$

Thus, the cardinality of `Students.address.city` is estimated at $|Students| \cdot 0.00867 = 34.75803$.

4.2 Select Sets and Selectivities

Select queries have the form $\tilde{\sigma}_p(A)$ over finite set A, where p is a predicate of one of the forms given in Figure 1. The cardinality of the result of the select query is $|A| \cdot sel(p, A)$. Though this section deals only with simple predicates, other papers (e.g., [SAC+79]) provide techniques for computing selectivities of simple predicates to estimate selectivities for more complex predicates.

Our primary contribution with respect to selectivity is to the extent table information to obtain better

Predicate form	Selectivity	Equivalent Selectivity	Estimation Formula
$a.\vec{\mathtt{f}} = c$	$sel(a.\vec{\mathtt{f}} = c, A)$	$sel(a' = c, A.\vec{\mathtt{f}})$	$\frac{1}{\lvert SKCS(A.\vec{\mathtt{f}}, C)\rvert}$
$a.\vec{\mathtt{f}} \in C$	$sel(a.\vec{\mathtt{f}} \in C, A)$	$sel(a' \in C, A.\vec{\mathtt{f}})$	$\frac{\lvert C\rvert}{\lvert SKCS(A.\vec{\mathtt{f}}, C)\rvert}$
$a.\vec{\mathtt{f}} = a.\vec{\mathtt{g}}$	$sel(a.\vec{\mathtt{f}} = a.\vec{\mathtt{g}}, A)$	$sel(a' = a'', A.\vec{\mathtt{f}} \times A.\vec{\mathtt{g}})$	$\frac{1}{\lvert SKCS(A.\vec{\mathtt{f}}, A.\vec{\mathtt{g}})\rvert}$
$a.\vec{\mathtt{f}} \in a.\vec{\mathtt{g}}$	$sel(a.\vec{\mathtt{f}} \in a.\vec{\mathtt{g}}, A)$	$sel(a' \in a'', A.\vec{\mathtt{f}} \times A.\vec{\mathtt{g}})$	$\frac{\lvert A.\vec{\mathtt{g}}\rvert}{\lvert SKCS(A.\vec{\mathtt{f}}, A.\vec{\mathtt{g}})\rvert}$

Table 3: Selectivity estimation formulae.

bounds on the domain of the attributes of the elements in a set, giving a better estimate for selectivity. We use the extent table information in two ways:

1. To get a better estimate on the number of values an attribute can take on a set using the techniques in the previous section to estimate the projectivity of the attribute.

2. To get a better estimate on the number of values that can be logically chosen to select against by looking up the projectivity of the attribute in the enclosing root type extent.

For two given sets A and B, with elements $a \in A$ and $b \in B$, let SKCS(A, B) be the smallest known common superset of the two sets (which may be the extent of the underlying type). We can derive the following formulae for use in estimating selectivity[7]:

$$\begin{aligned} |A \cap B| &\approx \frac{|A| \cdot |B|}{|\mathrm{SKCS}(A, B)|} \\ sel(a \in B, A) &\approx \frac{|B|}{|SKCS(A, B)|} \\ sel(a = b, A \times B) &\approx \frac{1}{|SKCS(A, B)|} \end{aligned}$$

4.2.1 Predicate selectivities

Table 3 summarizes the different forms of predicates defined in Table 1, and shows the formulae for estimating their selectivity. In this table, a is an element of the argument set A, c is some constant not computed from a taken from some set C, and C is some set not computed from elements of A. The first column in the table shows the form of the predicate, the second column the selectivity, the third column an approximately equal selectivity derived using the uniform distribution assumption, and the fourth column the resulting estimation formula using one of the equations from the previous section. Experiments with the derived formulas, reported in detail in [NCN95] show the promise of this approach.

[7] The complete derivations can be found in [NCN95].

Figure 5: Accuracy of estimates for mapping queries with different extent table disciplines.

5 Empirical Results

In order to evaluate the effectiveness of our estimates we implemented Perl scripts to create a sample database and to run mapping queries over it.[8] Our sample database contained 3500 households containing roughly 10000 people from 52 cities. Some of the data were uniform: names, street names, and zip codes. Others were not uniform: ages, states, city names (since the same city can be associated with several zip codes), and the number of residents per household. While we realize that this is a toy database, we observe that the data has many of the characteristics that make size estimation difficult.

Figure 5 shows the accuracy of our estimates for mapping queries. In this experiment, we ran mapping queries using all of the possible 118 path expressions of length ≤ 4 from our database. For each extent table discipline, we give the proportion of queries whose cardinalities were looked up directly, and estimated exactly, to within factors of 1.5, 2, and 4, and worse than a factor of 4. Because the queries were evenly distributed over the data, the "looked up" proportion indicates the relative size of the extent table.

The different extent table disciplines we used were the minimal extent table, level_1–level_3 with both the full extent table and with replacement when estimates differed by factors of 4, 2, and 1.5, and the full level_4 extent table. At level_4, all the queries were looked up directly in the extent table, since our queries had lengths 4 or less. As expected, for each level extent table, the accuracy of the estimates improves as the replacement factor μ decreases, with the full extent table providing the most accuracy. Also, as expected, level_i tables in general did better than level_{i-1} tables that used the same replacement factor.

Even with a minimal extent table, we were able to estimate more than half the queries within a factor of 4. As expected, increasing the sizes of a full extent table improves the quality of the results, with a

[8] We have, in fact, run selection queries over the test database as will, with similarly encouraging results.

dramatic improvement occurring between level_2 and level_3. In general, we found that we were able to estimate accurately path expressions of length n with an extent table of length $n-2$. Unfortunately, the size of the full level_2 extent table is quite high: approximately three times the size of the minimal extent table. We can get most of the accuracy improvement possible from a full level_n table using an adaptive level_n extent table, while keeping significantly less information. As a guide to deciding an appropriate discipline, the horizontal white line in the figure indicates twice the size of the minimal extent table.

6 Related Work

Most of the literature on cost models deals with relational or extended relational databases. System R [SAC+79] estimates selectivities for relational select, project and join operations using available indices. Where indices are not present, they use an arbitrary constant fraction. We instead use projectivities (which we can always approximate) to estimate selectivities of arbitrary path expressions. Also, they tend to over-estimate select set size because they assume a constant specified by the user is in the set of actual database values for the expression that they compare with the constant. Our approach more accurately assumes that the constant is taken over some logical underlying domain.

Statistical sampling and related techniques are frequently proposed for approximating selectivity and projectivity where the uniform distribution assumption is violated. Such approaches include Hou et.al. [HOD91], Lipton et.al. [LNS90], Haas and Swami [HS92], and Haas et.al [HNSS95]. Histogram techniques [PC84] are also used to improve selectivity estimates. Initial results combining statistical sampling techniques with our semantic estimation techniques is promising.

As an alternative to sampling, Sun et.al. use a regression model to approximate the underlying distribution of the data [SLRD93]. Antisampling techniques [Row85] are based on the premise that it is better to maintain a profile of all the information in the database, and deduce properties of the subsets of that data that are specified by a query from the information in the that profile. They assume that a subset is representative of its superset, and thus maintains the same data distributions.

With respect to optimization of queries over objects, little work has been done on cost and size modeling. At the query level, we only know of the work by Kemper and Moerkotte [KM90], which uses access support relations to derive sizes and cost of path expressions.

7 Conclusions

This paper addresses the problem of estimating cardinalities in persistent object bases. We observed that persistent object bases maintain natural set-subset relationships among the type extents of the types in the type hierarchy. We use this (and other) subset information to make more accurate cardinality estimations for query results. In this paper, we make the following contributions:

- We use *semantic* information to the schema manager (specifically inclusion relationships between sets) to infer cardinality estimates on the basis of known sizes of related sets.
- We propose a method for *adaptively* maintaining the information required for estimation in an extent table. We show empirically that an extent table need not be large but only intelligently maintained to allow for accurate cardinality estimation. In particular, keeping entries only for non-representative sample sets can be efficient and effective for estimating both image and select set cardinalities.
- We propose a more accurate method for estimating selectivities within the context of the uniform distribution assumption. This method uses image set cardinalities for root type extents to approximate the size of the domain over which users specify the values to select against.

Two areas of current and future work include better methods for estimating selectivities, and expanding this work to other complex object query operations. [NCN95] presents results for join and grouping queries.

References

[HNSS95] Peter J. Haas, Jeffrey Naughton, S. Sechadri, and Lynne Stokes. Sampling-based estimation of the number of distinct values of an attribute. In *Proceedings of the 21st VLDB Conference*, pages 311–322, 1995.

[HOD91] W.-C. Hou, G. Ozsoyoglu, and E. Dogdu. Error-constrained count query evaluation in a relational database. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 279–287. ACM, 1991.

[HS92] Peter J. Haas and Arun N. Swami. Sequential sampling procedures for query size estimation. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 341–350, 1992.

[KM90] Alfons Kemper and Guido Moerkotte. Access support in object bases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 364–374, 1990.

[LNS90] R. J. Lipton, J. F. Naughton, and D. A. Schneider. Practical selectivity estimation through adaptive sampling. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 1–11, 1990.

[NCN95] Marian H. Nodine, Mitch Cherniack, and Mark H. Nodine. Semantic cardinality estimation for object-oriented queries. Technical Report CS-95-35, Brown University Department of Computer Science, December 1995.

[PC84] Gregory Piatesky-Shapiro and Charles Cornell. Accurate estimation of the number of tuples satisfying a condition. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 256–276. ACM Press, 1984.

[Row85] Neil C. Rowe. Antisampling for estimation: An overview. *IEEE Transactions on Software Engineering*, SE-11(10):1081–1091, October 1985.

[SAC+79] P. Griffiths Selinger, M.M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 23–34, 1979.

[SLRD93] Wei Sun, Yibei Ling, Naphtali Rishe, and Yi Deng. An instant and accurate size estimation method for joins and selection in a retrieval-intensive environment. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 79–88. ACM, 1993.