How to Avoid Building DataBlades® That Know the Value of Everything and the Cost of Nothing*

Paul M. Aoki†
Computer Science Division, EECS Department
University of California, Berkeley, CA 94720-1776 USA
aoki@acm.org

Abstract

The object-relational database management system (ORDBMS) offers many potential benefits for scientific, multimedia and financial applications. However, work remains in the integration of domain-specific class libraries into ORDBMS query processing. A major problem is that the standard mechanisms for query selectivity estimation, taken from relational database systems, rely on properties specific to the standard data types; creation of new mechanisms remains extremely difficult because the software interfaces provided by vendors are relatively low-level. In this paper, we discuss extensions of the generalized search tree, or GiST, to support a higher-level but less type-specific approach. Specifically, we discuss the computation of selectivity estimates with confidence intervals using a variety of index-based approaches and present results from an experimental comparison of these methods with several estimators from the literature.

1. Introduction

The object-relational database management system (ORDBMS) offers many potential benefits for complex applications. Perhaps the most compelling is the ability to integrate domain-specific data-type libraries (which are termed extenders, data cartridges or DataBlades® by vendors) into the database engine; research ORDBMS applications have already been developed in scientific areas as diverse as bioinformatics [17], ocean and atmospheric sciences [12, 16] and high-energy physics [6]. In addition, commercial type libraries for ''mainstream'' but scientifically important types such as text, image, video, time series and geospatial data are now common.

This paper addresses a common problem area in the creation of such type libraries: the estimation of predicate *selectivity*, the fraction of records remaining after applying a selection predicate. Selectivity estimation code of reasonable quality is a significant performance issue because it integrates the application-specific operators into the database engine's query optimization infrastructure. Query optimizers compile declarative queries into *query plans*, dataflow programs that can be executed by the engine. To do so, optimizers require estimates of the execution costs of candidate (sub)programs. These cost estimates, based on formulæ that are largely parameterized by selectivity, need not be exact but must be sufficiently accurate for the optimizer to be able to avoid grossly inefficient query plans.

Selectivity estimation has been widely studied, but rarely in terms of a general framework for extensible database management systems. We describe a set of approaches based on a modification of the generalized search tree, or GiST [22], which supports flexible tree traversal [5]. Each approach uses approximate cardinality metadata, stored in the index nodes, to produce incrementally-refined selectivity estimates with confidence intervals. Although our approaches apply classic techniques (top-down tree traversal and random sampling), previous work in this area has been designed with different assumptions in mind or for different goals. As we will see, these differences motivate new algorithms.

From an engineering viewpoint, the main benefit of this index-based approach is that it applies a solution to a relatively well-understood problem (search) to a relatively poorly-understood problem (estimation). This enables database extenders, who are typically domain knowledge experts, to produce estimators without becoming experts in other domains (statistics, database cost models, *etc.*). The intuitive appeal of this approach is supported by an empirical trend observed by extensible database vendors: third-party extenders are far more likely to try to integrate search structures than they are to write non-trivial selectivity estimators.

---

* With apologies to Oscar Wilde (*Nowadays people know the price of everything and the value of nothing.*) and Alan Perlis (*A LISP programmer knows the value of everything, but the cost of nothing.*). DataBlade is a registered trademark of Informix Software.

† Author's current address: Xerox Palo Alto Research Center, 3333 Coyote Hill Rd., Palo Alto, CA 94304-1314 USA. Research at Berkeley supported by NASA under contracts FD-NAG5-6587 and FD-NAGW-5198.

From an algorithmic viewpoint, the theme of this work (which is closely related to work on sampling-based estimation, *e.g.*, [26]) is the ''best effort'' use of an explicit, limited I/O budget in the creation of interval estimates. It contains three main contributions. First, we provide a broad discussion of the ''GiST as histogram.'' We give a new algorithm that uses index traversal to produce selectivity estimates with deterministic confidence intervals (bounds with $p = 100\%$) over arbitrary user-defined types. Second, we consider the integration of tree traversal with index-assisted sampling (which produces confidence intervals with $p \leq 100\%$). Third, we provide results of an experimental comparative study (using a variety of geographic, Earth science and multimedia data sets) between our techniques and many of the proposed parametric multidimensional estimators. This is the only comparative study (concurrent work aside [1]) that compares these estimators to anything except the trivial estimator (based on the uniformity assumption).

The remainder of the paper is organized as follows. In Section 2, we provide a brief overview of the main background concepts and algorithms. We build on this background in Section 3, giving estimation algorithms based on index traversal and index-assisted sampling. Sections 4 and 5 explain our experimental infrastructure, procedures and results. Section 6 reviews related work. We conclude in Section 7. Additional algorithmic issues, experimental results and discussions of future work are contained in the full paper [4].

2. Background and assumptions

In this section, we briefly review the concepts and assumptions that underlie our approach. First, we give an overview of the approach. We then describe the specific index structure used. Specifically, we discuss the concepts of the generalized search tree and the pseudo-ranked tree.

Our approach to a general estimation infrastructure follows from two observations. First, a tree-structured index is a partitioning of an arbitrary data set at an arbitrary resolution. That is, the index recursively divides the indexed data into clusters; these clusters support efficient search, assuming that the data is *indexable* [24] and the index design is effective. Efficient indexed search over a given workload means that we examine a minimal number of extraneous objects over that workload. Second, in the process of implementing indices for their new data types, database extenders necessarily provide code to partition instances of the data types in question. By exploiting this existing code, we can solve our estimation problem ''for free'' (from the extender's point of view).

Symbol	Meaning
N	Number of leaf records.
h	Tree height (path length from root to leaf).
n	Number of samples.
c	True cardinality of a subtree ($\in [c^-, c^+]$).
c^0	Cardinality estimate center value ($\in [c^-, c^+]$).
c^-, c^+	Cardinality estimate {lower, upper} bounds.
u	Cardinality estimate uncertainty, or length of deterministic confidence interval.
$e.p$	Predicate (key) of node entry e.
$e.ptr$	Child pointer of node entry e.
c_e^0, c_e^-, c_e^+	c^0 (c^-, c^+) for a specific node entry, e.
$c_\Sigma^0, c_\Sigma^-, c_\Sigma^+$	Running c^0 (c^-, c^+).

Table 1. Summary of notation.

For an index-based approach to be practical, it should have limited maintenance overhead, controlled runtime overhead and controlled estimate precision. This paper explores how to perform index-based estimation in a way that permits us to strike a balance between these factors.

Table 1 summarizes the notation used in this paper.

2.1. Generalized search trees

Throughout this paper, we assume that indices are based on an extended version [5] of the basic GiST framework [22]. In this subsection, we briefly summarize the relevant properties of this extended framework.

The basic GiST generalizes the height-balanced, multiway tree. Each tree *node* contains a number of *node entries*, $e = <p, ptr>$, where each *predicate*, p, describes the subtree indicated by its corresponding disk pointer, ptr. The subtrees recursively partition the data records. However, they do not necessarily partition the data space. GiST can therefore model unordered, non-space-partitioning trees (*e.g.*, R-trees [19]) as well as ordered, space-partitioning trees (*e.g.*, B^+-trees).

The original GiST framework consists of a set of common *template* methods provided by GiST and a set of *extension* methods provided by the user. The template methods generally correspond to the functional interfaces specified in other access method interfaces: SEARCH, INSERT and DELETE. An additional method, ADJUSTKEYS, serves as a ''helper'' for INSERT and DELETE; this method enforces tree predicate invariants, *e.g.*, bounding-box containment for R-trees. The basic extension methods, which operate on predicates, include CONSISTENT, PENALTY and UNION. The novelty of GiST lies in the manner in which the behavior of the template methods is controlled (customized) by one or more of the extension methods. For example, SEARCH calls the CONSISTENT method to determine which subtrees it must visit (in a manner analogous to the usual B^+-tree key test); hence, only the user-defined

CONSISTENT must understand anything about the specific data type.

In recent work [5], we have added a number of additional extension methods. ACCURATE controls ADJUSTKEYS as CONSISTENT controls SEARCH, permitting ''sloppy'' predicates. PRIORITY allows SEARCH to traverse the tree in ways other than depth-first search. As each node is examined, a priority value is assigned to each of its CONSISTENT node entries; a priority queue is used to determine the order in which the descendant nodes are traversed. Finally, the iterator mechanism (consisting of three extension methods, STATEINIT, STATEITER and STATEFINAL) allow us to compute aggregate functions over the index records encountered during the index traversal.

To summarize, the original framework for defining tree-structured indices over arbitrary data types has been extended with a framework for flexibly traversing these indices and computing aggregate functions over the traversed nodes. In the remainder of the paper, we require these capabilities but no additional GiST properties.

2.2. Pseudo-ranked trees

Pseudo-ranked trees are one of the example applications enabled by the GiST extensions. Here, we define pseudo-ranking and explain its relevant properties. Much of this discussion and notation follows that of Antoshenkov [2].

An easy and intuitive way to construct a ''hierarchical histogram'' from a tree index is to augment every non-leaf node entry with a cardinality count (*i.e.*, the total number of leaf records in the specified subtree). Such counts are commonly called *ranks* [38]. Inserting or deleting a record results in node modifications from leaf to root because any such update changes the cardinality of every subtree containing that record. This is generally considered to be impractical in a production DBMS (though bulk-update, common in data warehouses, can reduce this cost). A lower-cost alternative is to compute upper bounds on the corresponding subtree's cardinality using a node's height within the tree and simple fanout statistics [37]. Such bounds may be imprecise if the tree is not full.

Pseudo-ranking balances the cost of rank maintenance against bound imprecision. The amortized space and time costs of pseudo-ranking are low enough that it has been incorporated in a high-performance commercial DBMS, Rdb/VMS (now Oracle Rdb).

Definition 1 [2]: *A tree is **pseudo-ranked** if, for each node entry e, we can compute a cardinality estimate, c_e^0, as well as lower and upper bounds, $c_e^- \le c_e^0 \le c_e^+$, for the subtree indicated by $e.ptr$. Also, $c_e^- \le c \le c_e^+$, where c is the true cardinality. (If $c_e^- = c_e^0 = c_e^+$, the tree is **ranked**.)*

We do not yet specify how c_e^+ and c_e^- are computed, though (by convention) we assume that $c_e^- = c_e^0 = c_e^+ = 1$ if e is a leaf record. Additionally, we will assume that the index satifies the following condition:

Definition 2 [2]: *Let i indicate a given node entry, and let $child(i)$ indicate the node to which $i.ptr$ refers. A pseudo-ranked tree satisfies the **nested bound condition** if $c_i^+ \ge \sum_{j \in child(i)} c_j^+$ and $c_i^- \le \sum_{j \in child(i)} c_j^-$ for all non-leaf index records i.*

In other words, the interval $[c_i^-, c_i^+]$ in the parent node entry always contains the aggregated intervals of its child node entries. The extended GiST framework can enforce these bounds automatically using the ACCURATE extension method.

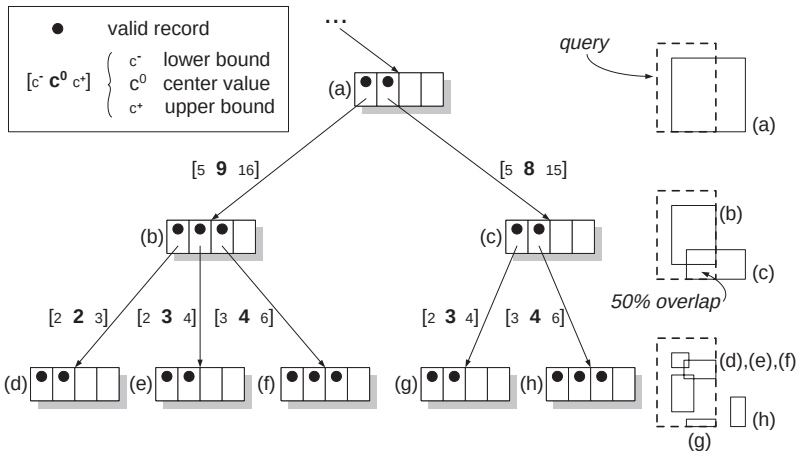
Figure 1(a) shows a pseudo-ranked GiST. For clarity, the predicates in this example are simple 2D bounding rectangles, so the tree is structurally an R-tree. The (explicitly stored) cardinality estimate c^0 for each node entry appears next to its pointer, along with the corresponding (derived) values for the bounds c^- and c^+. Antoshenkov [2] gives an example formula for c^- and c^+, $\prod_{h-1}^{i=0}(1 + A \cdot Q^i) - 1$, that provides fixed imprecision bounds for a given tree height. The values for c^- and c^+ in Figure 1(a) assume the use of this example formula with $A = ½$ and $Q = ½$. (Figure 1(b) will be discussed in Section 3.1.)

Use as hierarchical histogram. Descending the tree results in interval estimates of non-decreasing tightness. First, node entries that are not CONSISTENT can be pruned. Second, because of the nested bound condition, an interval estimate computed from a parent node entry can never be tighter than the interval estimate computed from any of its (aggregated) descendant node entries.

As a ''sloppy histogram,'' the pseudo-ranked tree has precision that depends on the acceptable update overhead. In Sections 3.3 and 5.4 of the full paper, we explain why the imprecision caused by pseudo-ranking is easily computed and small in practice.[1]

Use in sampling. In the remainder of the paper, we assume that index-assisted sampling (hereafter, *index sampling*) is implemented using acceptance/rejection (A/R) sampling applied to pseudo-ranked trees. A detailed discussion of this method is beyond the scope of this paper and may be found elsewhere [2, 37]. The

[1] As an aside, note also that we can actually tune the tree ''online'' by increasing the bounding formula dynamically (*i.e.*, all aspects of pseudo-ranking continue to work without modification); hence, we can always start with a ranked tree and increase the imprecision until we reach an acceptable level of update overhead.



(a) Tree structure with pseudo-ranks and predicates.

node	u	c^-_Σ	c^0_Σ	c^+_Σ	u_Σ
(a)	-	5	**13**	31	26
(c)	15	7	**12**	20	13
(b)	11	9	**12**	17	8
(f)	3	9	**11**	14	5
(g)	2	7	**10**	12	3
(e)	2	9	**9**	10	1
(d)	1	9	**9**	9	0

Notes:

- Node (h) is pruned immediately after node (c) is visited.
- Ties between nodes with equal u values are broken arbitrarily.

(b) Traversal and results.

Figure 1. A pseudo-ranked GiST with an example traversal.

relevant points are that the index supports equiprobable sampling[2] with an additional cost that is proportional to the error introduced by pseudo-ranking.[3]

3. Algorithms

We now apply the techniques of Section 2 to produce new estimation algorithms. First, we describe a new algorithm based on index traversal. We then give a novel way to combine this traversal mechanism with index-assisted random sampling.

3.1. Interval estimation using traversal

In this subsection, we propose a method of index traversal for estimation. We first describe how the method works. We then discuss the relative disadvantages of other, more ''obvious,'' solutions. Finally, we provide more detail and work through an example.

The high-level problem statement is that we wish to probe the tree index, examining nodes to find node entries whose predicates match (*i.e.*, are CONSISTENT with) the query. However, unlike a search algorithm, the desired estimation algorithm need not obtain records from the leaf level. Instead, the cardinality intervals associated with the CONSISTENT node entries are aggregated into an approximate query result size. The reason to limit the number of nodes visited is cost, both in terms of CPU and I/O; most importantly, I/Os used during query optimization compete for disk arm time with actual query processing.

Obtaining precise answers means examination of nodes that maximize the reduction of *uncertainty*, $u_\Sigma = c^+_\Sigma - c^-_\Sigma$ (the summations are over the nodes examined thus far). We cannot tell in advance how much each node will reduce our uncertainty, so we cannot construct an optimal on-line algorithm; we must settle for a heuristic that can use the information available in each node entry to guess how much following its pointer will reduce the overall uncertainty. Fortunately, it turns out that we can never ''lose'' precision by descending a pointer — descending a pointer from node entry e to $child(e)$ never increases uncertainty if the index satisfies the nested bound condition (see Appendix C of the full paper) — so we have a great deal of flexibility in choosing a traversal order.

A general approach using prioritized traversal [5] and incremental aggregation is outlined in Figure 2. The approach gives us three major degrees of freedom: how to prioritize (order) the tree traversal, how to aggregate (*i.e.*, what statistics to keep) and how to stop traversal.

Previous approaches fit within this framework but do not work well in practice. *Depth-first* (PRIORITY = node depth) and *breadth-first* (PRIORITY = node height) traversal algorithms have two main flaws. First, they do not limit the number of nodes visited. For example, the ''key range

[2] A/R sampling from an unbalanced pseudo-ranked tree (and therefore, by immediate extension, from a pseudo-ranked forest) returns each record with equal probability (see Appendix D of the full paper). We will need this result in the next section because sampling from the frontier of a traversal is essentially sampling from a forest.

[3] Pseudo-ranking implies that the exact size of the population being sampled is unknown. The simplest way of dealing with this is to perform random sampling from a subpopulation, or domain of study [13, Sec. 2.13]; this simply introduces a relative error proportional to the overestimate of population size.

```
1.  push root into priority queue
2.  initialize statistics
3.  while (not done)
4.    pop node entry e from priority queue
5.    update statistics (subtracting e)
6.    fetch node n using e.ptr
7.    for each CONSISTENT node entry e′ ∈ n
8.      push e′ into priority queue
9.      update statistics (adding e′)
```

Figure 2. Basic algorithm outline.

Scale Factor	Mean Query Selec. (%)	Queries Having Root as Split Level (%)	
		R-tree	B^+-tree
1	0.041	83	0.24
2	0.85	91	6.4
3	31	99	93

Table 2. Split-level and dimensionality.

estimator'' in some versions of DB2/400 (which even limits the depth of tree descent) sometimes read-locks the entire index for minutes. This causes severe problems for update transactions.[4] Second, neither depth-first nor breadth-first traversal are well-suited for incremental use. If we cut off traversal after visiting some number of nodes, depth-first will have wasted much of its effort traversing low-level, low-uncertainty regions (*e.g.*, leaf nodes), while breadth-first may have visited the children of nodes that were fully subsumed by the query (and therefore had low uncertainty). The *split-level* heuristic [3], described in more detail in Appendix B of the full paper, also uses PRIORITY = node depth. By stopping descent when the query predicate is CONSISTENT with more than one node entry in the current node, it visits at most $\log N$ nodes. This heuristic is effective for low-dimensional partitioning trees (*e.g.*, B^+-trees), but as the data type becomes more complex, the split-level heuristic degrades into an examination of the root node. Table 2 illustrates this effect using data sets and queries drawn from our experiments in Section 4. The table shows the percentage of queries (out of 10,000 drawn from each of three different selectivity ranges) that stop after inspecting the root node of a given image feature vector index. The first column shows what happens if all 20 dimensions are indexed, while the second describes the corresponding values if only the highest-variance dimension is indexed. (The key size is kept the same to produce structurally similar trees.)

We plainly require incremental algorithms, of which the following prioritized traversal variant is an example. References to ''line i'' are to the corresponding lines in Figure 2.

Priority. PRIORITY is computed from the uncertainty, u, of each node entry. Hence, node entries with high uncertainty are traversed first (line 4).

Statistics. The algorithm keeps three running statistics: c_Σ^-, c_Σ^0 and c_Σ^+. As the algorithm descends the tree, node entries with CONSISTENT predicates are pushed into the priority queue and their associated statistics are added to each statistic (lines 8,9). As node entries are removed from the priority queue, their statistics are also removed (lines 4,5).

If the node entry predicate is wholly contained (subsumed) by the query predicate, the running statistics are updated using the c^-, c^0 and c^+ values stored in the node entry. However, if the node entry predicate only overlaps the query predicate, the statistic c_Σ^- is not changed (as if $c^- = 0$) — this is a lower bound and we cannot guarantee that any records beneath this node entry are actually CONSISTENT with the query predicate. Similar reasoning applies to the estimate c^0 (but not the upper bound c^+).

For example, the R-tree traversal estimators depicted in Figure 1(b) use a trivial overlap-based estimator. Given a non-leaf index entry, we estimate c^0 by assuming that the number of records matching a query is proportional to the fraction of the entry's predicate that overlaps the query. This is analogous to the logic used in unidimensional histogram estimation. If the extender does not provide the logic to compute such domain-specific local estimators, a generic estimator for c^0 can be used (*e.g.*, assuming that half of the records match).

Stopping rule. The algorithm may halt (line 3) when the reduction of uncertainty ''tails off'' or some predetermined limit on the number of nodes is reached.[5] An obvious way to measure tail-off is to track the rate of change of the confidence interval width, halting when a discontinuity is reached. We defer additional discussion until Section 5.

To show the algorithm in operation, we return to Figure 1. Figure 1(b) shows an example of the prioritized traversal algorithm running to completion on the pseudo-ranked tree depicted in Figure 1(a). The nodes are visited

[4] This is due solely to the length of the index traversal process: ''When large files are involved (usually a million records or more), an estimate key range can take seconds or even minutes to complete'' [27].

[5] A decision-theoretic framework [42] might well be possible, but it is not immediately clear how to compute expected utility in a query optimization context.

in an order corresponding to the uncertainty, u, of their corresponding parent node entries. Most of the iterations are straightforward except for those relating to nodes (a) and (c). When node (a) is examined, we find that the query predicate covers (exactly) half of node (c)'s node entry predicate. Hence, when the algorithm examines node (a), $c^-_{(c)} = 0$ and $c^0_{(c)} = 4$ (instead of 5 and 8, respectively). In addition, when node (c) is examined, node (h) is pruned (its predicate is not CONSISTENT with the query predicate).

3.2. Sampling as a supplement to traversal

Random sampling does not produce deterministic confidence intervals as does our traversal-based estimator. However, it can potentially produce tighter interval estimates with high confidence. The main issue is how to decide between when to apply each of the techniques. In this subsection, we first discuss some dualities and synergies between the two techniques. These intuitions lead us to a strategy that switches from traversal to sampling.

Index traversal alone might not permit us to meet our desired accuracy goals. Indices cluster records together in a way that minimizes the number of nodes that must be retrieved to answer a query and work well when the index "fits" the query — records should be *dense* with respect to the queries that use the index. If the data is *sparse*, *i.e.*, matching records are scattered among many nodes, sampling may work better.

The problem of deciding when to switch to sampling is closely related to the problem of halting traversal (discussed in the previous subsection). The main difference is that our decision process must somehow model the expected benefit of sampling. A simple strategy is to base this decision on conservative (Chernoff/Hoeffding) confidence intervals. As traversal proceeds, we compare the most recent decrease in confidence interval width to the corresponding decrease for a conservative sampling estimator. (Put another way, we compare the *measured* slopes of the traversal estimator's confidence intervals and the *predicted* slopes of the sampling confidence intervals.) This works because the conservative confidence intervals decrease deterministically as more samples are obtained. Switching to sampling makes sense when traversal has begun to produce results worse than the expected results from sampling.

4. Experimental procedure

In the next two sections, we describe a set of experiments we conducted to assess the effectiveness of our techniques. In this section, we discuss the indices and algorithms used as well as the data/query sets on which they were used. Additionally, we detail the estimation algorithms we used as benchmarks. Finally, we sketch the overall experimental design.

In the introduction, we emphasized the generality of an index-based approach to selectivity estimation. We selected multidimensional point data for these comparative experiments for two main reasons. First, multidimensional data has many applications in scientific and multimedia databases. Second, there are several proposed selectivity estimators with which we can compare our results. (By contrast, had we chosen set data indexed by RD-trees [22], we could only have compared our techniques with random sampling.)

4.1. Data structures and algorithms

This subsection describes our specific implementations of the general algorithms described in Section 3. We discuss the index structures, the index loading algorithms, and the estimation algorithms in turn.

Indices. We implemented pseudo-ranked GiSTs using libgist 1.0.[6] Since our experimental application domain is multidimensional, we used a GiST based on bounding rectangles (*i.e.*, an R-tree). To avoid conflating the effects of pseudo-ranking with the effects of other experimental variables, we measured only ranked trees. (As mentioned in Section 2, the worst-case effect of pseudo-ranking on our interval estimates has easily-computed bounds.)

Loading algorithm. Loading has a strong effect on the effectiveness of an index. We used a variety of loading algorithms, each of which represented a class of related algorithms: insertion-load using randomly-ordered records, insertion-load using Hilbert-clustered [29] records, bulk-load using Hilbert-clustered records, bulk-load using STR-clustered [33] records.

Estimators. The traversal and aggregation interfaces allow us to implement estimation using prioritized traversal, breadth-first traversal, and A/R index sampling in about 500 lines of C++. These extensions are admittedly somewhat tricky, since each essentially implements a specialized state machine; however, this is not much of an issue because the extender plugs code into these extensions rather than writing new ones.

For both base-table (simple random) sampling and index (A/R) sampling, we implemented a variety of running interval estimators for the mean. These estimators

[6] libgist, including driver programs and a suite of predefined access methods, consists of about 20K lines of C++ and is freely available from http://gist.cs.berkeley.edu/. libgist 1.0 implements primary access methods (data records stored in the leaf nodes) on top of a simple storage manager that can be replaced by the SHORE recoverable storage manager [9] at compile-time.

Data set	Records	Dimensionality			Density			
					Insertion		Bulk-load	
		D	D_0	D_2	random	Hilbert	Hilbert	STR
GNIS	1,517,114	2	1.837	1.746	3.997	2.773	2.274	1.627
Uni2			2	2	31.33	4.894	2.974	2.553
GTSPP	1,167,671	4	1.906	0.9368	10.79	3.361	4.276	1.950
Uni4			4	4	171.8	11.28	7.573	4.477
Blob	26,021	20	5.101	1.235	0.06804	0.06948	0.1904	0.07910
Uni20			20	20	62.66	12.82	7.924	7.430

Table 3. Data set characteristics.

were based on conservative [23], central limit theorem (CLT) [20], and non-parametric BC_a bootstrap confidence intervals [14]. Conservative techniques are more appropriate than those based on CLTs for the sample sizes under study but provide weaker bounds; in terms of useful sample sizes, we have empirically observed that the non-parametric BC_a bootstrap falls somewhere in between the other two.

4.2. Bases for comparison

As our ''benchmarks,'' we selected several parametric point estimators from the literature on spatial databases. Different estimators apply to *random-centered* and *object-centered* window queries [39].[7] For random-centered queries, we implemented and compared estimators based on the uniformity assumption, the Hausdorff fractal dimension D_0 [15] and density (expected stabbing number) [43]. For object-centered queries, we also used an estimator based on the correlation fractal dimension D_2 [8].

We chose not to compare our techniques with non-parametric estimators based on space-partitioning for a simple reason: these techniques require summary data that is exponential in the embedding dimension, D. For example, a simple histogram-like variant of the density technique [43] would have required $3^D \approx 3.5$ billion density points for $D = 20$. The same argument applies to space-partitioning multidimensional histograms [40, Ch. 9]. (When details and implementations become available, comparisons with more parsimonious non-parametric methods such as wavelet-encoded histograms [35] should be instructive.)

[7] Random-centered queries establish the base location (*e.g.*, center point) of a query shape from a probability distribution defined on the underlying space. Object-centered queries select a random object from the data set to establish the query location. For queries over highly skewed data sets, object-centered queries tend to have higher selectivities.

4.3. Data sets

We chose to conduct experiments on selected data sets rather than on synthetic data sets generated using simple parametric distributions. The latter approach does enable sequences of experiments to be conducted using the distribution parameters as a controlled variable. However, the usual simple distributions (*e.g.*, normal, Zipf) do not tend to describe the distribution of real data sets particularly well, which makes the benefits somewhat moot.

We used three separate real data sets of varying embedding dimensionality, D:

- Geographic coordinates from the USGS GNIS data set ($D = 2$) [44]. This represents GIS workloads.
- Spatial coordinates plus time from the NOAA GTSPP data set ($D = 4$) [21]. GTSPP is a bathythermograph (ocean temperature) database and represents Earth science workloads.
- Image feature vectors from the Berkeley Digital Library Project's Blobworld system ($D = 20$) [10]. The 20 dimensions result from applying the singular value decomposition to 256-bin histogram values in the CIE LUV color space and then truncating. This represents multimedia workloads.

For each real data set, we also generated uniform random data sets of the same dimensionality and cardinality. Uniform data has two specific properties of interest. First, it often represents a kind of ''worst case'' for simple clustering techniques because it reduces the effectiveness of partitioning heuristics. We will see an example of this in Table 5. Second, it represents a kind of ''best case'' for most parametric estimators — uniform data is simply a degenerate case of most models.

Table 3 summarizes the characteristics of each data set. We measured the fractal dimensions D_0 and D_2 of each real data set using the software of Belussi and Faloutsos [8]. (The fractal dimension of the uniform random data sets was not measured as it is equal to the embedding dimension.) The full paper contains some additional observations about the characteristics of our data sets.

4.4. Experimental design

Our experimental design varied data sets, load algorithms, the use of random-centered *vs.* object-centered queries, query scale factors and query aspect ratios (the last two factors are defined below). After a pilot study of 500 replications (queries) per experiment, we performed 10,000 replications per experiment.

Queries. For each data set, we generated (uniform) random-centered and object-centered query rectangles. We generated the queries in three distinct equivalence classes, or *scale factors*. All members of a query class have identical geometries and can therefore be expected to have a result size similar (but not identical) to that of its fellow class members. Each member of a class is considered equally likely (*i.e.*, is assigned equal weight in the results).

The extent (side lengths) of the query rectangles varied depending on the data set applications. For example, the GTSPP query parameters were based on our experience with a particular geoscience application [16]. We also varied the geometric *aspect ratio* by doubling the length of the basic query shape along one axis. The actual query shape parameters can be found in Appendix E of the full paper, though the resulting selectivity factors (which are much more critical to the final results) are specified in Section 5.

Criteria. Loosely, we define ''success'' in our estimation problem as 1% absolute error for queries with selectivity of 1% or less. If the query is less selective, then we have no Boolean criterion analogous to the 1% ''success''; we simply seek more precise answers. We justify this approach as follows. An application for histograms is unclustered index scan selection. Here, we wish to know the number of records so we can compute the number of expected base-table I/Os — crudely, the unclustered index ''break-even'' point is when the selectivity equals the inverse of the mean index node occupancy (fill factor). In practice, this is ''a single-digit percentage'' [31]. For this problem, it is therefore most important to know whether or not the selection result size falls into the ''1%'' category.[8]

In spite of our use of query classes, the arithmetic mean of the absolute errors tends to be misleading due to skew and/or bimodality. Wherever we present a mean absolute error, we also present the maximum error as a measure of disperson. To summarize the relative error within each class, we use a ratio-of-sums (weighted harmonic) mean (*cf.* [30, Ch. 12], [40, p. 73]).

[8] Few vendors show interest in single-table estimation with resolution much finer than 1%. Most vendors currently use equidepth histograms or quantile values, for which the number of buckets corresponds directly to the maximum absolute error. Vendor tuning documents generally recommend fewer than 100 buckets [4].

5. Experimental results

We now present the results of the experiments just described. First, we summarize the effectiveness of the various alternative estimators. Second, we discuss some results on the use of heuristics for limiting the duration of index traversal. Note that the results presented represent points in the potential application space for the technique. It would plainly be impossible to demonstrate high effectiveness for *all* indices (after all, the effectiveness of the technique depends on the effectiveness of the index itself).

Discussion of the hybrid traversal/sampling approach described in Section 3.2 is deferred to the full paper due to space limitations.

5.1. Estimation effectiveness

Each estimator was instrumented to produce incremental results as it processed between 1 and 64 index nodes. The results in this subsection describe a ''snapshot'' of the results at 12 nodes (*i.e.*, a fixed, identical cost for both traversal and sampling). This is a somewhat arbitrary number, chosen because it is 3-4 times the height of the index; this affords a few probes from root-to-leaf so that sampling has some chance of working.

Tables 4 and 5 show two illustrative extracts from this snapshot (full tables are contained in Appendix A of the full paper). We do not present results from the Hilbert-order data sets because their behavior is qualitatively similar to that of STR. For similar reasons, we do not present the results of the varied aspect ratios. (The differences in both cases can be quantitatively significant, even interesting, but are not illustrative for the points of study.) Results using breadth-first traversal are also excluded because it was dominated by uncertainty-prioritized traversal.

The format of Tables 4 and 5 is as follows. For each combination of data set, query scale factor and query center type (object or uniform random), we give the mean selectivity and a variety of error metrics for each applicable estimator. Since some estimators produce different results depending on how the index was loaded (insertion- or bulk-loaded), we list these separately. Here, ''index'' indicates prioritized index traversal, whereas ''sample'' indicates simple random sampling with replacement from a base table (no index) in conjunction with conservative

Data / Scale / Query	Mean Sel. (%)	s.d.	Load	Est	Mean Rel. Error	Mean Abs. Error (%)	Max. Abs Error (%)	Mean c.i. Width (%)	Max. c.i. Width (%)
GNIS / 2 / o	0.2085	0.0014	i	index	0.0957	0.0199	0.2410	0.5123	5.7478
			b	index	0.0176	0.0037	0.0795	0.1110	0.5893
				sample	1.3324	0.2778	7.8113	35.5364	44.0060
				D_2	0.5793	0.1208	1.1034		
GNIS / 3 / o	12.9348	0.0643	i	index	0.0476	0.6159	2.7284	13.2880	33.5049
			b	index	0.0766	0.9903	4.3601	11.1679	25.6842
				sample	0.4964	6.4206	42.1618	47.9793	70.6604
				D_2	0.5683	7.3514	21.4943		
GTSPP / 2 / o	0.2099	0.0002	i	index	0.8553	0.1795	2.4018	49.3148	72.0985
			b	index	0.4989	0.1047	1.6321	1.7342	15.5191
				sample	0.4165	0.0874	2.0259	35.5396	39.7573
				D_2	0.8379	0.1759	2.4385		
GTSPP / 3 / o	11.6502	0.0324	i	index	0.3165	3.6874	12.3467	68.6537	81.2857
			b	index	0.1393	1.6228	6.4663	27.9042	42.3499
				sample	0.0618	0.7205	4.1933	46.9762	61.8927
				D_2	0.9738	11.3447	24.3291		
Blob / 2 / o	0.8536	0.0004	i	index	0.9869	0.8424	4.3740	34.8422	84.3703
			b	index	0.4674	0.3990	2.5149	7.3323	37.1930
				sample	0.5157	0.4402	6.2682	36.1739	45.5452
				D_2	3.3550	2.8640	3.7072		
Blob / 3 / o	31.2809	0.0437	i	index	0.7035	22.0057	45.2617	85.9696	98.1131
			b	index	0.4205	13.1550	30.1561	64.7500	94.6313
				sample	0.1391	4.3505	25.6728	61.7832	70.6604
				D_2	0.4368	13.6644	35.3293		

Table 4. Excerpted results: real data — 12 nodes.

Data / Scale / Query	Mean Sel. (%)	s.d.	Load	Est	Mean Rel. Error	Mean Abs. Error (%)	Max. Abs Error (%)	Mean c.i. Width (%)	Max. c.i. Width (%)
Uni4 / 2 / u	0.0019	0.0000	i	index	0.3139	0.0006	0.0035	44.9318	82.7869
			b	index	0.0630	0.0001	0.0013	0.0271	0.2098
				sample	1.9370	0.0037	0.1716	35.3320	35.5038
				uniform	0.9891	0.0019	0.0042		
				D_0	9.0136	0.0171	0.0189		
			i	density	268.6663	0.5102	0.5120		
			b	density	23.4731	0.0446	0.0464		
Uni4 / 3 / u	4.3201	0.0226	i	index	0.1729	0.7469	4.0015	77.0006	95.9137
			b	index	0.0264	0.1140	0.6523	16.4926	33.1527
				sample	0.1093	0.4724	2.6366	39.6432	45.5732
				uniform	0.9809	4.2375	8.2187		
				D_0	0.3398	1.4681	4.9760		
			i	density	4.4314	19.1441	22.7537		
			b	density	0.3881	1.6765	4.7865		

Table 5. Excerpted results: synthetic data — 12 nodes.

confidence intervals.[9] Note that the confidence interval widths are full widths, not half-widths.[10]

The tables reveal several points that agree with intuition or previously-established results. First, the estimate error is generally best for the index-traversal method with bulk-loaded data. The success of the bulk-loaded index estimator is to be expected since it is using relatively well-partitioned data. The relative ineffectiveness of the

[9] The fractional conservative confidence interval widths vary between scale factors, which may be confusing since (by definition) they are fixed for a given number of samples. However, we truncate each query's interval widths at $[0, N]$ (*i.e.*, 0% and 100%); hence, the mean width for each scale factor varies based on edge effects, *i.e.*, how many of its constituent intervals happen to "stick out" beyond these limits.

[10] This is for two reasons. First, the deterministic confidence intervals are usually asymmetric. Second, the conservative confidence intervals we present are actually asymmetric as well. This is because we truncate the low/high ends of the probabilistic intervals at the lowest/highest known deterministic intervals, whether obtained from a previous index traversal phase or the trivial bounds $[0, N]$.

insertion-loaded index estimation implies that the traversal-based technique is best used with an index optimized for the workload. Second, the estimate error and confidence interval widths degrade for all methods as the dimensionality of the data increases. We expect an effect like this due to the ''curse of dimensionality,'' and analogous degradations have been widely documented elsewhere in the statistics and computer science literature [41]; the index-based technique works better with lower-rank projections of the same data set. Third, random sampling performs quite poorly unless the selectivity is large (*i.e.*, $\gg 1\%$). This is due to a combination of the small sample size with (1) the weakness of conservative confidence intervals and (2) the small selectivity (''needle in a haystack'') effect.

There are a number of more interesting results as well. First, the uniformity, density and Hausdorff fractal dimension (D_0) estimators were all quite unstable. The best results are obtained when the data and query characteristics match the model assumptions (*e.g.*, lower dimensionality, approximately hypercubic queries, more uniform data); the density estimator is particularly hard-hit by the poor insertion-load R-tree quality. However, given that these estimators simplify away nearly all characteristics of the data set and query, wide variation in accuracy seems inevitable. Second, the correlation fractal dimension (D_2) estimator performs quite well given that it does not look at the data and the query sets at all. It does not perform as well as originally reported [8], but this is to be expected given that the queries are non-equilateral. Third, while the index traversal estimator with bulk-loaded data generally and most consistently produces reasonable confidence intervals (recall our ''1%'' success criterion), none of the estimators is particularly successful on high-dimensional, uniformly-distributed data sets.

5.2. Effectiveness of traversal limit heuristics

The previous subsection described our experiments for a fixed, heuristic traversal limit of 12 index nodes. In Section 3.2, we discussed the possibility of more elaborate heuristics. For example, a moment's thought would lead one to expect a roughly geometric drop-off in the incremental gain (decrease in confidence interval width) from each node — as one descends to a new level, having exhausted the previous level, the expected number of index records covered by any given node entry drops by a constant factor (*i.e.*, by the fanout).

Investigation reveals that the confidence interval width curves do tend to drop off as expected. Our prioritized traversal algorithm, by its heuristic ordering of nodes by uncertainty, tends to smooth out the curve; nodes that would cause ''jagged'' or step-function behavior under simple breadth-first search tend to be processed earlier. Specifically, we found the confidence interval widths produced by prioritized traversal algorithm dominated those of breadth-first traversal in all test cases. For similar reasons, pseudo-ranking tends to smooth the curve as well.

We did implement and measure traversal heuristics that detected when the decrease in confidence interval width ''tailed off,'' *i.e.*, that stopped when the rate of change reached a local change-point. However, we observed the following effect: even with the smoothing factors just described, the curves tended to have *several* change-points, usually corresponding to the transition from one level to the next lower level. In the tests that use ranked trees, the individual graphs appear almost piecewise-linear. This is not an issue for shorter trees (two or three levels), but tends to cause the algorithm to stop earlier than desired for taller trees (four levels or more). Though a more sophisticated variant may be useful, we cannot recommend the naive tail-off heuristic for general use.

6. Related work

In this section, we briefly survey the relevant database literature. Specifically, we discuss non-parametric selectivity estimation techniques (which relate to tree traversal), estimation using random sampling, and tree condensation. We refer the reader to Mannino's survey [34] for background information about selectivity estimation; the references given here are generally incremental with respect to that survey. Many additional references are given in the full paper.

Extensible estimation methods: The current state of the art is to provide black-box user hooks for selectivity estimation functions [28]. That is, the extender is told to write a user-defined function that computes the selectivity of any clause containing an instance of their user-defined type.

Non-parametric statistics: Non-parametric density estimation encompasses a variety of techniques. The approaches discussed below have one or two of the following major disadvantages. First, they are difficult to apply in a ''generic'' extensible framework because they all require a mapping from the given data type to some D-dimensional numeric representation. While some mapping is *always* possible, a mapping into a representation with a ''nice'' distribution (one that does not require an inordinate amount of summary data to be approximated with reasonable error) is necessarily domain-dependent and may prove difficult to find — from the point of view of the extender, this may trade one complex and unnatural task for another. (By constrast, using index structures for estimation takes advantage of mappings that the extender has already created and optimized.) Second, space-

partitioning schemes require storage exponential in D.

• **Model-fitting techniques**. Methods based on regression, wavelets and neural nets [11, 32, 35] have been used to summarize attribute frequency distributions. The proposed techniques have some additional disadvantages. First, like the parametric estimators discussed in this paper, they are all point estimators and provide no interval bounds. Second, with a few exceptions, the ability to perform dynamic updates of the summary data is limited.

• **Histograms**. Now well-established [40], conventional histograms rely on space-partitioning schemes. Various forms of indexed main-memory multidimensional histograms have also been proposed [36]. Secondary memory histograms and hierarchical estimation are not considered in this work; neither are the problems of space-partitioning.

• **Index-assisted statistics**. Several researchers have noted that balanced tree structures can be viewed as a hierarchy of (approximately) equidepth histograms [3]. Others have used access methods to compute aggregate [18] or density [25] functions. This work does not generally trade off precision against cost.

Tree traversal: An enormous literature exists on heuristic tree search in artificial intelligence. Much of this work deals with handling complexity (*e.g.*, variable ordering, high levels of redundancy) that does not arise in our context. However, the priority-based algorithm of Section 3 could be described as a type of best-first search. The related database literature is discussed more thoroughly in Section 3 and Appendix B; again, unlike previous work, the methods described in this paper make the precision/cost tradeoff explicit.

Sampling: Database sampling takes many forms and has many applications; for more information, we recommend the surveys by Olken [38] and Haas [7, Sec. 9]. The most specifically-relevant index-assisted sampling literature has been summarized in Section 3.

7. Conclusions and future directions

In this paper, we have argued that indexing techniques form the basis for a general and practical approach to selectivity estimation in extensible databases. The software base is not large and the user-defined extension methods are not difficult to write. Most importantly, this approach allows us to take advantage of the expertise and development effort of third-party database extenders; aside from random sampling, which is completely general but relatively costly, other proposed techniques are either specific to particular data types or require additional integration effort for non-multidimensional data types.

We have described why we found previously proposed methods for index-assisted estimation to be unsatisfactory in conjunction with generalized search trees (as opposed to, *e.g.*, B^+-trees). We then introduced incremental traversal algorithms and adaptive traversal/sampling algorithms to address these problems.

The paper contains several additional, smaller contributions. We have provided experimental evidence that the traversal-based technique presented in Section 3 can provide selectivity estimates with lower error than the parametric estimators in the literature. (This need not have been the case, since the technique is based on traversal of structures designed for search rather than estimation.) We have noted the importance of interval estimates for certain query optimization decisions, and have shown that our traversal technique can meet or come close to meeting our goal of 1% error for 1%-selectivity queries in practice. This is contrast to the proposed parametric estimators, which provide only point estimates. Finally, we have illustrated that several of the proposed parametric estimators can be unstable under reasonable conditions.

There are many possible areas for additional work. These are summarized in the full paper; perhaps the most significant lies in the investigation of formal performance bounds, perhaps arising from the ongoing work on the theory of indexability [24].

References

[1] S. Acharya, V. Poosala and S. Ramaswamy, "Selectivity Estimation in Spatial Databases," *Proc. 1999 SIGMOD*, Philadelphia, PA, May 1999, to appear.

[2] G. Antoshenkov, "Random Sampling from Pseudo-Ranked B^+ Trees," *Proc. 18th VLDB*, Vancouver, BC, Canada, Aug. 1992, 375-382.

[3] G. Antoshenkov, "Dynamic Query Optimization in Rdb/VMS," *Proc. 9th ICDE*, Vienna, Austria, Apr. 1993, 538-547.

[4] P.M. Aoki, "Algorithms for Index-Assisted Selectivity Estimation," Tech. Rep. UCB//CSD-98-1021, Univ. of California, Berkeley, CA, Oct. 1998.

[5] P.M. Aoki, "Generalizing ''Search'' in Generalized Search Trees," *Proc. 14th ICDE*, Orlando, FL, Feb. 1998, 380-389.

[6] M. Athanas, "(Object) Relational Databases for High Energy Physics Data," *Proc. 7th Int'l Conf. on Computing in High Energy Physics*, Berlin, Germany, Apr. 1997, C365. http://www.ifh.de/CHEP97/.

[7] D. Barbará, W. DuMouchel, C. Faloutsos, P.J. Haas, J.M. Hellerstein, Y. Ioannidis, H.V. Jagadish, T. Johnson, R. Ng, V. Poosala, K.A. Ross and K.C. Sevcik, "The New Jersey Data Reduction Report," *IEEE Data Eng. Bull. 20*, 4 (Dec. 1997), 3-45.

[8] A. Belussi and C. Faloutsos, "Estimating the Selectivity of Spatial Queries Using the 'Correlation' Fractal Dimension," *Proc. 21st VLDB*, Zürich, Switzerland, Sep. 1995, 299-310.

[9] M.J. Carey, D.J. DeWitt, M.J. Franklin, N.E. Hall, M.L. McAuliffe, J.F. Naughton, D.T. Schuh, M.H. Solomon, C.K. Tan, O.G. Tsatalos, S.J. White and M.J. Zwilling, "Shoring Up Persistent Applications," *Proc. 1994 SIGMOD*, Minneapolis, MN, May 1994, 383-394.

[10] C. Carson, S. Belongie, H. Greenspan and J. Malik, "Region-Based Image Querying," *Proc. IEEE Wksp. on Content-Based Access of Image & Video Libraries*, San Juan, Puerto Rico, June 1997, 42-49.

[11] C.M. Chen and N. Roussopoulos, "Adaptive Selectivity Estimation Using Query Feedback," *Proc. 1994 SIGMOD*, Minneapolis, MN, May 1994, 161-172.

[12] Y. Chi, C.R. Mechoso, M. Stonebraker, K. Sklower, R. Troy, R.R. Muntz and E. Mesrobian, "ESMDIS: Earth System Model Data Information System," *Proc. 9th SSDBM*, Olympia, WA, Aug. 1997, 116-118.

[13] W.G. Cochran, *Sampling Techniques (3rd Ed.)*, John Wiley & Sons, New York, 1977.

[14] T.J. DiCiccio and B. Efron, "Bootstrap Confidence Intervals," *Stat. Sci. 11*, 3 (Aug. 1996), 189-228.

[15] C. Faloutsos and I. Kamel, "Beyond Uniformity and Independence: Analysis of R-trees Using the Concept of Fractal Dimension," *Proc. 13th PODS*, Minneapolis, MN, May 1994, 4-13.

[16] W.E. Farrell, J. Gaffney, J. Given, R.D. Jenkins and N. Hall, "A Hydrographic Database Built on Montage and S-PLUS," Sequoia 2000 Tech. Rep. 94/47, Univ. of California, Berkeley, CA, Mar. 1994.

[17] D.J. Flanders, S. Weng, F.X. Petel and J.M. Cherry, "AtDB, the Arabidopsis Thaliana Database, and Graphical-Web-Display of Progress by the Arabidopsis Genome Initiative," *Nucl. Acids Res. 26*, 1 (Jan. 1, 1998), 80-84.

[18] S. Ghosh, "SIAM: Statistics Information Access Method," in *Proc. 3rd SSDBM* (Luxembourg, July 1986), R. Cubitt, B. Cooper and G. Özsoyoğlu (eds.), EUROSTAT, Luxembourg, 1986, 286-293.

[19] A. Guttman, "R-trees: A Dynamic Index Structure for Spatial Searching," *Proc. 1984 SIGMOD*, Boston, MA, June 1984, 47-57.

[20] P.J. Haas, "Large-Sample and Deterministic Confidence Intervals for Online Aggregation," *Proc. 9th SSDBM*, Olympia, WA, Aug. 1997, 51-62.

[21] D. Hamilton, "GTSPP Builds an Ocean Temperature-Salinity Database," *Earth Sys. Monitor 4*, 4 (June 1994), 4-5. http://www.nodc.noaa.gov/GTSPP/gtspptxt-w52.html.

[22] J.M. Hellerstein, J.F. Naughton and A. Pfeffer, "Generalized Search Trees for Database Systems," *Proc. 21st VLDB*, Zürich, Switzerland, Sep. 1995, 562-573.

[23] J.M. Hellerstein, P.J. Haas and H. Wang, "Online Aggregation," *Proc. 1997 SIGMOD*, Tucson, AZ, May 1997, 171-182.

[24] J.M. Hellerstein, E. Koutsoupias and C.H. Papadimitriou, "On the Analysis of Indexing Schemes," *Proc. 16th PODS*, Tucson, AZ, May 1997, 249-256.

[25] H. Hinterberger, "Multidimensional Data Visualization Design Tradeoffs: Speed vs. Detail," in *Proc. 3rd SSDBM* (Luxembourg, July 1986), R. Cubitt, B. Cooper and G. Özsoyoğlu (eds.), EUROSTAT, Luxembourg, 1986, 85-97.

[26] W.-C. Hou, G. Özsoyoğlu and B.K. Taneja, "Processing Aggregate Relational Queries with Hard Time Constraints," *Proc. 1989 SIGMOD*, Portland, OR, May 1989, 68-77.

[27] "Query Optimizer Seize," Technical Document 6511134, IBM AS/400 Division, Rochester, MN, Aug. 1997. http://as400service.rochester.ibm.com/.

[28] "Guide to the Virtual-Table Interface, Version 9.01," Part No. 000-3692, Informix Corp., Menlo Park, CA, Jan. 1997.

[29] H.V. Jagadish, "Linear Clustering of Objects with Multiple Attributes," *Proc. 1990 SIGMOD*, Atlantic City, NJ, May 1990, 332-342.

[30] R. Jain, *The Art of Computer Systems Performance Analysis*, John Wiley & Sons, New York, 1991.

[31] J. Kirkwood, *Sybase SQL Server 11: An Administrator's Guide*, Thomson Comp. Press, Boston, 1996.

[32] S. Lakshmi and S. Zhou, "Selectivity Estimation in Extensible Databases - A Neural Network Approach," *Proc. 24th VLDB*, New York City, Aug. 1998, 623-627.

[33] S.T. Leutenegger, M.A. López and J.M. Edgington, "STR: A Simple and Efficient Algorithm for R-tree Packing," *Proc. 13th ICDE*, Birmingham, England, Apr. 1997, 497-506.

[34] M.V. Mannino, P. Chu and T. Sager, "Statistical Profile Estimation in Database Systems," *Computing Surveys 20*, 3 (Sep. 1988), 191-221.

[35] Y. Matias, J.S. Vitter and M. Wang, "Wavelet-Based Histograms for Selectivity Estimation," *Proc. 1998 SIGMOD*, Seattle, WA, May 1998, 448-459.

[36] M. Muralikrishna and D.J. DeWitt, "Equi-depth Histograms for Estimating Selectivity Factors for Multi-Dimensional Queries," *Proc. 1988 SIGMOD*, Chicago, IL, June 1988, 28-36.

[37] F. Olken and D. Rotem, "Random Sampling from B^+ Trees," *Proc. 15th VLDB*, Amsterdam, the Netherlands, Aug. 1989, 269-277.

[38] F. Olken, *Random Sampling from Databases*, Ph.D. dissertation, Univ. of California, Berkeley, CA, 1993.

[39] B.-U. Pagel, H.-W. Six, H. Toben and P. Widmayer, "Toward an Analysis of Range Query Performance in Spatial Data Structures," *Proc. 12th PODS*, Washington, DC, May 1993, 214-221.

[40] V. Poosala, *Histogram-Based Estimation Techniques in Database Systems*, Ph.D. dissertation, Univ. of Wisconsin, Madison, WI, 1997. UMI No. 9716074.

[41] D.W. Scott, *Multivariate Density Estimation*, John Wiley & Sons, New York, 1992.

[42] K.D. Seppi, J.W. Barnes and C.N. Morris, "A Bayesian Approach to Database Query Optimization," *ORSA J. Comp. 5*, 4 (Fall 1993), 410-419.

[43] Y. Theodoridis and T. Sellis, "A Model for the Prediction of R-tree Performance," *Proc. 15th PODS*, Montreal, Québec, Canada, June 1996, 161-171.

[44] U. S. Geological Survey, "Geographic Names Information System," Data Users Guide 6 (4th printing, revised), U. S. Department of the Interior, Reston, VA, 1995.