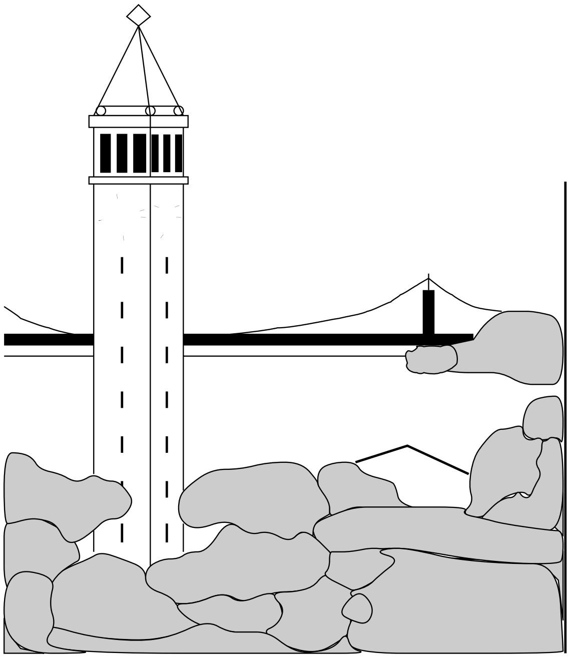
Algorithms for Index-Assisted Selectivity Estimation

Paul M. Aoki



Report No. UCB//CSD-98-1021

October 1998

Computer Science Division (EECS)
University of California
Berkeley, California 94720

Algorithms for Index-Assisted Selectivity Estimation

Paul M. Aoki[†]

Department of Electrical Engineering and Computer Sciences
University of California
Berkeley, CA 94720-1776

Abstract

The standard mechanisms for query selectivity estimation used in relational database systems rely on properties specific to the attribute types. For example, histograms and quantile values rely on the ability to define a well-ordering over the attribute values. The query optimizer in an object-relational database system will, in general, be unable to exploit these mechanisms for user-defined types, requiring the user to create entirely new estimation mechanisms. Worse, writing the selectivity estimation routines is 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 user-defined selectivity estimation in a variety of ways. We discuss the computation of selectivity estimates with confidence intervals over arbitrary data types using indices, give methods for combining this technique with random sampling, and present results from an experimental comparison of these methods with several estimators from the literature.

1. Introduction

Relational query optimizers compile declarative queries into *query plans*, dataflow programs that can be executed efficiently. To perform their combinatorial calculations, optimizers rely on simple statistical cost models to predict the performance of candidate (sub)programs. Specifically, they require estimates of the CPU and I/O execution cost of specified relational algebra expressions. These include individual algebraic operators (*e.g.*, the relational selection operator applied to a table) as well as more complex expressions (*e.g.*, a sequence of joins and selections). While the estimates need not be exact, they must be sufficiently accurate for the optimizer to be able to eliminate grossly inefficient programs from consideration.

In this paper, we focus on the most basic estimation problem: predicting predicate *selectivity*, the fraction of records remaining after applying a selection predicate to a single table. The general problem, while fundamental, has not been widely addressed for extensible database management systems. We argue that the selectivities of certain classes of predicates are best determined by probing a suitable index structure. Specifically, we describe a set of approaches based on a modification of the generalized search tree, or GiST [HELL95], which allows for flexible tree traversal [AOKI98a].

From an engineering viewpoint, the main benefit of an 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 in areas such as computer vision, to produce

[†] Research supported by NSF under grant IRI-9400773 and by NASA under grants FD-NAG5-6587 and FD-NAGW-5198.

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 generate selectivity estimators.

From an algorithmic viewpoint, the theme of this work (which is closely related to that of the work on sampling-based estimation) is the ‘‘best effort’’ use of an explicit, limited I/O budget in the creation of interval estimates. It contains four main contributions. First, we provide a broad discussion of the ‘‘GiST as histogram.’’ We give a new algorithm for the use of index traversal to produce selectivity estimates with deterministic confidence intervals over arbitrary user-defined types. Second, we consider the integration of tree traversal with index-assisted sampling. This strategy permits *dynamic condensation*, a technique which we introduce here in the context of search trees. Third, we demonstrate that integrated traversal and sampling can improve the interval estimates produced by either technique alone. Fourth, we provide results of an experimental comparative study between our techniques and all of the proposed multidimensional parametric estimators (*i.e.*, those based on Hausdorff fractal dimension [FALO94], correlation fractal dimension [BELU95] and density [THEO96]). To our knowledge, this is the only comparative study that compares any of these spatial estimators to anything except the trivial estimator (based on the uniformity assumption).

The paper is organized as follows. In the remainder of the introduction, we discuss (at a high level) the issues involved in using indices for selectivity estimation. In Section 2, we provide a brief overview of some 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 the related work. We conclude and suggest future work in Section 7.

1.1. Desiderata for selectivity estimation

In this subsection, we provide a more systematic motivation for our work. First, we list the important criteria for a selectivity estimation method. Second, we provide a brief description of the ways in which index-assisted estimation already meets these criteria. Finally, we describe the areas requiring further development — these being the areas that will be addressed in the remainder of the paper.

We consider each of the following items to be critical goals for any proposed selectivity estimation method in an object-relational database management system (ORDBMS):

- Support for user-defined types. Such types include non-numeric types such as set data and image feature vectors.
- Ease of integration. As discussed above, extensive expertise in database query processing technology should not be required.
- Limited maintenance overhead. If the estimation method involves metadata or other forms of preprocessing, the steady state cost (in terms of both CPU and I/O) should be reasonable.
- Limited runtime overhead. When invoked, the estimator should not unduly increase latency or interfere with system throughput (*e.g.*, due to processing cost, locking behavior, *etc*).
- Estimate precision. The requirements for precision depend on the decision problem at hand; one such requirement will be discussed in Section 4.

Broadly speaking, index-assisted selectivity estimation meets each of the criteria listed above.

- Many of the estimation techniques proposed in the literature may be difficult or impossible to apply in the case of arbitrary user-defined types and operators. (Additional discussion of this point may be found in Section 6.) By contrast, we observe that a tree index is a partitioning of an *arbitrary data set* at an *arbitrary resolution*. That is, it recursively divides the indexed data into clusters that support efficient search (assuming that the index's design is effective). This partitioning leads naturally to very general solutions to our problem, as will be seen in Section 3.
- 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. If the selectivity estimation algorithm can be made (largely) independent of the data type, we have solved our problem ‘‘for free.’’
- An index must be built and maintained. This is not unreasonable; database administrators are accustomed to building indices over frequently-queried columns, both for search performance and for improved metadata. Some vendors even recommend ‘‘at least one index per table’’ [IBM98].
- If our estimation method traverses the index top-down, we can stop whenever we like, obtaining information at any resolution desired.
- A similar argument applies to estimate precision.

The main question raised by the preceding discussion lies in the inherent tradeoff between estimation runtime cost and estimation precision. This paper explores how to perform index-based estimation in a way that permits us to strike this balance.

2. Background

In this section, we briefly review some underlying concepts that define the type of index structure we use and the

Symbol	Meaning
N	Number of leaf records.
$\hat{N}$	Estimated number of leaf records ($\geq N$).
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 deterministic confidence interval.
$\tilde{c}$	Cardinality estimate error.
$e.\,p$	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.
$c_\Sigma^0, c_\Sigma^-, c_\Sigma^+$	Cumulative c^0 (c^-, c^+).

Table 1. Summary of notation.

characteristics of the various operations we perform. We first discuss the relevant aspects of generalized search trees. We then review the definition and properties of pseudo-ranked trees.

Table 1 summarizes the notation used in this paper.

2.1. Generalized search trees

Throughout this paper, we assume that indices are based on the GiST framework [HELL95] as extended in [AOKI98a]. In this subsection, we briefly summarize the relevant properties of this framework.

GiST generalizes the notion of a 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 ptr. The subtrees recursively partition the data records. However, they do not necessarily partition the data space. GiST can therefore model ordered, space-partitioning trees (*e.g.*, B^+-trees [COME79]) as well as unordered, non-space-partitioning trees (*e.g.*, R-trees [GUTT84]).

The original GiST framework of [HELL95] consists of (1) a set of common *internal* methods provided by GiST and (2) a set of *extension* methods provided by the user. The internal methods generally correspond to the functional interfaces specified in other access method interfaces: SEARCH, INSERT and DELETE. An additional internal method, ADJUSTKEYS, serves as a ‘‘helper function’’ for INSERT and DELETE. This method enforces tree predicate invariants, such as 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 generic internal methods is controlled (customized) by one or more of the extension methods. For example, CONSISTENT and PENALTY control SEARCH and INSERT, respectively.

In [AOKI98a], we added a number of additional extension methods which will be relevant to this paper. ACCURATE controls ADJUSTKEYS as CONSISTENT controls SEARCH. PRIORITY allows SEARCH to traverse the tree in ways other than depth-first search. Finally, the iterator mechanism (consisting of three extension methods, STATEINIT, STATEITER and STATEFINAL, that are loosely based on Illustra’s user-defined aggregate function interface [ILLU95]) allow us to compute functions over the index records encountered during the index traversal.

To summarize, [HELL95] defines a framework for defining tree-structured indices over arbitrary data types, and [AOKI98a] provides a framework for flexibly traversing these indices and computing an aggregate function over the traversed nodes. In the remainder of the paper, we assume we have these capabilities (but do not require additional GiST properties).

2.2. Pseudo-ranked trees

Pseudo-ranked trees are one of the example applications enabled by the GiST extensions of [AOKI98a]. Here, we define pseudo-ranking and explain its relevant properties. Much of this discussion and notation follows

[ANTO92].

Definition 1 [ANTO92]: *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 $e.ptr$. (If $c_e^- = c_e^0 = c_e^+$, the tree is simply said to be **ranked**.)*

We do not yet specify how c_e^+ and c_e^- are computed. However, by convention, we assume that:

- $c_e^- = c_e^0 = c_e^+ = 1$ if e is a leaf record.
- The overall estimate of tree cardinality is defined by $\hat{N} = \sum_{e \in root} c_e^+ \ge N$.

Practically any tree access method can be pseudo-ranked without any modifications to the underlying data structure. For example, if we can determine a given node's height within the tree, we can use simple fanout statistics to compute (crude) bounds on the cardinality of the subtrees to which its node entries point.

Additionally, we will assume that the index satifies the following condition:

Definition 2 [ANTO92]: *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.*

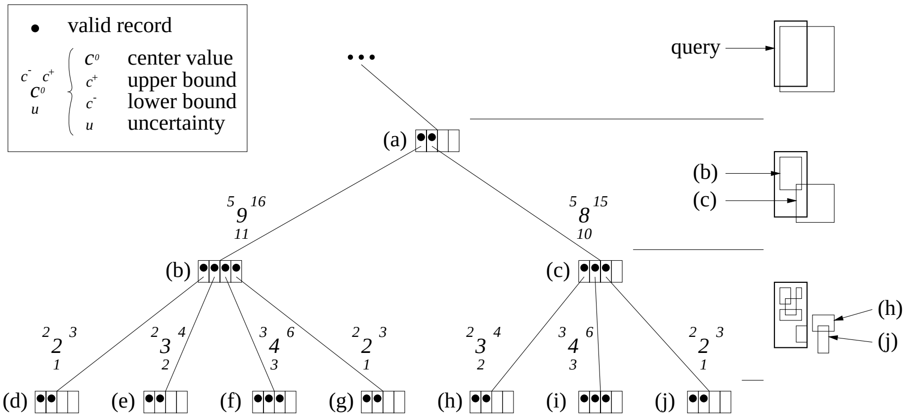
In other words, the interval in the parent node entry always contains the aggregated intervals of its child node entries.

As mentioned just above, we can compute weak bounds using only height and fanout statistics. However, much better bounds can be obtained by storing c^0 values in each node entry. c^- and c^+ can also be stored explicitly, or they can be derived as fixed ε-bounds from c^0. For historical reasons, we will refer to stored c^0 values as *ranks*.[1]

Figure 1(a) shows a pseudo-ranked 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^+.[2] (Figure 1(b) will be discussed in Section 3.1.)

[1] What we describe here is essentially a partial-sum tree [KNUT75, WONG80]. The term ‘‘ranks’’ arose from the use of cardinality information in cumulative partial-sum trees [KNUT73] and comes to the database literature via [OLKE89].

[2] In Figure 1(a), the values for c^- and c^+ are computed using the example formula from [ANTO92] using parameter values $A = ½$ and $Q = ½$.



(a) R-tree and predicates.

node	c^-_Σ	c^0_Σ	c^+_Σ	u_Σ	$\tilde{c}$
(a)	10	**17**	31	26	5
(c)	8	**13**	22	14	1
(b)	12	**15**	22	10	3
(f)	12	**14**	19	7	2
(i)	12	**13**	16	4	1
(e)	12	**12**	14	2	0
(h)			*pruned*		
(d)	12	**12**	13	1	0
(g)	12	**12**	12	0	0
(j)			*pruned*		

(b) Traversal and results.

Figure 1. Prioritized traversal in a pseudo-ranked R-tree.

The goal of this infrastructure is to reduce the cost of maintaining the ranks. Every insertion or deletion in a (precisely) ranked tree results in a leaf-to-root update because the update changes the cardinality of every subtree containing that record. This is, in general, impractical in a production DBMS (though many databases are, in fact, bulk-updated). By contrast, the amortized space and time costs of pseudo-ranking are low enough that it has been incorporated in a high-performance commercial DBMS, Oracle Rdb [ANTO92, SMIT96]. In this paper, we will assume that the index has been tuned so that these update costs are acceptably low. The tradeoff is that some amount of imprecision is added to the estimate.

How much imprecision does pseudo-ranking actually add? The point of [ANTO92] is that it depends entirely on the degree of update overhead we are willing to accept; the example formula used to compute c^- and c^+ in [ANTO92] provides fixed imprecision bounds for a given tree height. We discuss this further in Sections 3.3 and 5.4. Note also that we can actually tune the tree by increasing the bounds dynamically (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. (And if we never update the index after an initial bulk-load, we need never accept any imprecision!)

3. Algorithms

We now apply the results and 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. Finally, we discuss the effects of pseudo-ranking on the precision of these estimation algorithms.

3.1. Interval estimation using traversal

In this subsection, we propose a new 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 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 are CONSISTENT with the query. However, unlike a search algorithm, the desired estimation algorithm need not descend to the leaf level. Instead, the cardinality intervals associated with the CONSISTENT node entries are aggregated into an approximate query result size. This explains our interest in pseudo-ranking: better pseudo-ranked bounds result in better estimates.

Our goal, then, is an algorithm that visits the nodes such that we maximize the reduction of *uncertainty*, $u = c^+_\Sigma - c^-_\Sigma$. Pseudo-ranking is not the only source of uncertainty. Notice in Figure 1(a) that node (b) is completely subsumed by the query, whereas (c) is not. If we have only visited node (a), we know that the number of records under node (b) that overlap the query must lie in the range $[5, 16]$. However, the number of records under node (c) actually lies in $[0, 15]$ rather than $[c^-, c^+] = [5, 15]$ because we do not know how many records actually lie outside of the query rectangle.

Since we cannot tell in advance how much each node will reduce our uncertainty, we cannot construct an optimal on-line algorithm; instead, 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 for a proof.) This gives us a great deal of freedom in designing our traversal algorithm. We therefore propose a tree traversal framework that uses a simple *priority-based* traversal algorithm [AOKI98a]. The algorithm, an example of which is given below, descends the tree starting from the root and follows links with the highest priority (*i.e.*, uncertainty). Node entries with CONSISTENT predicates are pushed into a priority queue; at any given time, the sum of the intervals in the priority queue is our running interval estimate of the query result size. The algorithm may halt when the reduction of uncertainty ‘‘tails off’’ or some predetermined limit on the number of nodes is reached. 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.

At this point, it is reasonable to ask: why not use depth-first or breadth-first search? First, as search algorithms, they fail to place limits on the number of nodes visited. This is true even if we ‘‘fix’’ our costs by only visiting nodes down to some specified level in the tree. For example, the ‘‘key range estimator’’ in some versions of DB2/400 essentially examines all nodes above the leaves; the estimator sometimes read-locks the entire index for minutes, causing severe problems for update transactions [IBM97a].[3] Second, neither search algorithm is well-suited for incremental use. If we cut off search after visiting some number of nodes, depth-first search will have wasted much of its effort traversing low-uncertainty regions (*e.g.*, leaves), while breadth-first may have visited the children of many nodes that were fully subsumed with the query (and therefore had low uncertainty).

The so-called split-level heuristic [ANTO93] is somewhat less risky. This heuristic, described in more depth in Appendix B, descends until the query predicate is CONSISTENT with more than one node entry in a given node and then stops. It therefore visits between 1 and $\log N$ nodes. This heuristic is effective for low-dimensional partitioning trees (*e.g.*, B^+-trees) but turns out to be very sensitive to the data type and index structure. 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 scales) 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.) As the data type becomes more complex, the split-level heuristic degrades into an examination of the root node, which is not very informative in general.

The bottom line is that an estimation algorithm based solely on simple structural considerations (*e.g.*, level) or data-driven considerations (*e.g.*, intersections between the query and the node entry predicates) may run much too long or stop much too soon. We plainly require incremental algorithms, of which the prioritized traversal algorithm is an example.

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 R-tree depicted in Figure 1(a). Each node entry

[3] 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.’’

		Queries Having Root as Split Level (%)	
Scale Factor	Mean Query Selec. (%)	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.

stores a predicate (bounding box), a link to its child node, and a cardinality count for the subtree rooted at that child node. Note that 12 out of the 16 records match the query predicate (which completely contains leaf nodes (d), (e), (f), (g) and (i)).

The following interesting points may be observed about the traversal depicted in Figure 1. First, as stated above, the uncertainty never increases; in fact, u_Σ happens to decrease with each link followed. Second, observe that the fact that uncertainty is non-increasing does not imply that the cardinality estimate c_Σ^0 becomes strictly more accurate (equivalently, that c_e never increases). The absolute error $\hat{c}$ does, in fact, diverge temporarily from the true cardinality after visiting node (b).

3.2. Sampling as a supplement to traversal

Random sampling does not produce deterministic confidence intervals as does our traversal-based estimator. However, it has the potential to produce much 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. We then provide a conservative heuristic for switching.

Index traversal alone might not permit us to meet our desired accuracy goals. Consider the following intuition. Indices work well for search (and, by extension, for traversal-based selectivity estimation) when they cluster records together in a way that minimizes the number of nodes that must be retrieved to answer a query. That is, the index must ‘‘fit’’ the query — records should be *dense* with respect to the queries that use the index. Indexing works poorly when the data is *sparse*, *i.e.*, matching records are scattered among many nodes.

Observe that the respective cases described above complement those in which random sampling works poorly (well). For example, cluster sampling works best when the data is scattered most widely, since this reduces the sample variance (as well as the number of expected I/Os required to find non-zero samples). If we detect that traversal is performing poorly, *i.e.*, that our interval estimate is not shrinking quickly, we may find it worthwhile to switch to sampling.

As a side effect of switching to sampling after traversal, the upper levels of the tree will have been condensed. That is, we need not start each sample probe from the root node. Instead, we can do better by sampling from the *frontier* of the traversal; this frontier is already materialized in the form of the traversal priority queue. Depending on how far traversal progressed and the effectiveness of buffering, this might save many I/Os. Even in a large-buffer environment in which I/O is less of a factor, we save the CPU costs of repetitively reprocessing the upper-level nodes (which may be considerable for more complex data types). Note that this kind of *dynamic condensation* customizes its benefits to the specific query, as opposed to a static condensation policy that (*e.g.*) simply condenses the top few levels.

Obviously, 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 confidence intervals [HOEF63]. As traversal proceeds, we compare the most recent decrease in confidence interval width to a conservative approximation of 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 hypothetical slopes of the sampling confidence intervals.) Switching makes sense when traversal has begun to produce results worse than the expected results from sampling.

In the remainder of the paper, we assume that index-assisted 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; such discussions may be found in [ANTO92, OLKE89] or Appendix D.

3.3. On the effects of pseudo-ranking

As previously noted, pseudo-ranking introduces a degree of imprecision into our estimates. In this subsection, we discuss the degree to which this affects our results.

Our main observation is that, aside from the rank values themselves, pseudo-ranked and (precisely) ranked trees built over the same data set are *identical*. Conceptually, the net effect of pseudo-ranking on both traversal estimation and sampling estimation is to ‘‘expand’’ the tree by a number of ‘‘ghost’’ records (ones that do not match any predicate). If we use the techniques of [ANTO92], this expansion is bounded by a constant, ε, which is known *a priori*.

The main effect on the traversal-based estimator is to shift the interval endpoints. The shift is bounded by an amount proportional to the pseudo-ranking imprecision at the root node, ε. For example, the confidence interval width is increased by a total of $(c^{-}_{\Sigma}/\varepsilon) + (c^{+}_{\Sigma} \cdot \varepsilon) \leq 2 \cdot c^{+}_{\Sigma} \cdot \varepsilon$. Hence, pseudo-ranking introduces *relative* error dependent on ε, *not* absolute error dependent on ε.

Some minor technical issues that must be resolved before we can use A/R record sampling for estimation on pseudo-ranked trees. First, in Appendix D, we show that 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. We need this result because sampling from the frontier of the traversal is essentially sampling from a forest. Second, we observe that pseudo-ranking implies that the exact size of the population being sampled is not known. The simplest way of dealing with this is to perform simple random sampling from a subpopulation, or domain of study [COCH77, Sec. 2.13]; again, this simply introduces a relative error proportional to the overestimate of population size.

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. For these comparative experiments, we selected multidimensional point data because there are several alternative selectivity estimators in the literature with which we can compare our results. By contrast, had we chosen set data (perhaps indexed using RD-trees [HELL95]), 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. All experiments used an implementation of pseudo-ranked R-trees as described in Section 3. To avoid conflating the effects of pseudo-ranking with the effects of other experimental variables, we measured only (precisely) ranked R-trees. As discussed in Section 3, the worst-case effect of pseudo-ranking on our interval estimates has easily-computed bounds. The ranked case also arises after bulk-loading or bulk-update.

We implemented pseudo-ranked R-trees using libgist 1.0. 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 [CARE94] at compile-time.

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. We randomized the records and then loaded the index by repeated insertion. Our R-tree implementation uses Guttman's quadratic node splitting algorithm.[4]

- Linearized insertion-load. As above, but the data set is sorted in Hilbert curve order as a data-clustering heuristic [JAGA90] before insertion.
- Linearized bulk-load. We used Hilbert curve order as the page-packing heuristic [KAME93].
- Non-linearized bulk-load. We used STR [LEUT97] because of its simplicity.

Estimators. The traversal and aggregation interfaces of [AOKI98a] allow us to implement estimation using prioritized traversal, breadth-first or level-at-a-time traversal (*à la* [ANDE88, WHAN94]), and acceptance/rejection sampling in about 500 lines of code. These extensions are admittedly somewhat tricky, since each essentially implements a variety of specialized state machines; however, this is not much of an issue because the extender plugs code into these extensions rather than writing new ones.

For the R-tree traversal estimators, we used a trivial overlap-based estimator. That is, given an non-leaf index entry, we make a uniformity assumption: the number of records matching a query is proportional to the fraction of the entry's predicate volume that overlaps the query. This is analogous to the logic used in, *e.g.*, unidimensional histogram estimation.

For both heap and index sampling, we implemented a variety of running interval estimators for the mean. These estimators were based on conservative [HELL97a], central limit theorem (CLT) [HAAS97], and non-parametric $\mathrm{BC_a}$ bootstrap confidence intervals [DICI96]. 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 $\mathrm{BC_a}$ 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 [PAGE93].[5] For random-centered queries, we implemented and studied estimators based on the uniformity assumption, the Hausdorff fractal dimension D_0 [FALO94] and density (expected stabbing number) [THEO96]. For object-centered queries, we also

[4] An optimal $O(n^D)$ algorithm exists, where n is the number of node entries in a node and D is the embedding dimension [GARC98]. However, it is plainly impractical for some of our data sets (*e.g.*, those with $n \geq 30$ and $D = 20$).

[5] Random-centered queries establish the location (*e.g.*, center) of a query 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. Hence, object-centered queries always have nonzero selectivity. Furthermore, for queries over highly skewed data sets, object-centered queries will tend to have higher selectivities than random-centered queries.

used an estimator based on the correlation fractal dimension D_2 [BELU95].

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, [THEO96] recommends using a simple histogram-like variant of the density technique for non-uniform data. However, this approach is plainly impractical in our context — the suggested gridding method would have required $3^D \approx$ 3.5 billion density points for $D = 20$. The same argument applies to space-partitioning multidimensional histograms [POOS97, Ch. 9]. (When details and implementations become available, comparisons with more parsimonious non-parametric methods such as wavelet-encoded histograms [MATI98] should be instructive.)

4.3. Data Sets

We explicitly 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. The problem is that the usual distributions (*e.g.*, normal, Zipf) do not describe the spatial distribution of real data sets particularly well. We usually cannot measure a given real data set to determine these distribution parameters and then use the parameters to make performance predictions, which limits the usefulness of such experimental sequences.

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

- Geographic coordinates from the USGS GNIS data set ($D = 2$) [USGS95]. This is a “national” version of the Sequoia 2000 storage benchmark [STON93].
- Spatial coordinates plus time from NOAA’s GTSPP data set ($D = 4$) [HAMI94].
- Image feature vectors from the Berkeley Digital Library Project’s Blobworld system ($D = 20$) [CARS97]. The 20 dimensions result from applying the singular value decomposition to 256-bin histogram values in the CIE LUV

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.

color space and then truncating.

For each real data set, we also generated uniform random data sets of the same dimensionality and cardinality. Uniform data is not ‘‘yet another’’ form of synthetic data as it has some specific properties of interest. First, uniform data 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 Mathematica scripts described in [BELU95]. (The generalized fractal dimension of the uniform random data sets was not measured as it is equal to the embedding dimension.)

A few general observations on Table 3 are in order here. First, subunitary density values, such as that seen in the Blobworld data, can be explained by skew. Skew causes gaps, which in turn results in (relatively) large regions that are not covered by any predicate. Second, the high density values in the ‘‘random’’ column (which translate into high overlap) show clearly that insertion-loaded R-trees do not cluster data very well, particularly in high dimensions. Third, the relatively high density values for bulk-loaded uniform data are an artifact of the specific bulk-loading algorithms (which are primarily concerned with the properties of the leaf page predicates rather than those of the higher-level predicates).

4.4. Queries and Evaluation Criteria

For each data set, we generated (uniform) random-centered and object-centered query rectangles. To reduce measurement variance, 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 based on the data set applications. For example, the GTSPP query parameters were based on our experience with the geoscience application described in [FARR94]. We also varied the geometric *aspect ratio* by doubling the length of the basic query shape along one axis.

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.* [JAIN91, Ch. 12], [POOS97, p. 73]).

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. Consider that the most important application for histograms is (arguably) unclustered index scan selection. Here, we wish to know the number of records so we can compute the number of expected heap 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’’ [KIRK96]. For this problem, it is therefore most important to know whether or not the selection result size falls into the ‘‘1%’’ category.[6]

4.5. Experimental Design

We used a full-factorial experimental design. That is, we varied data sets, load algorithms, query scale factors, query aspect ratios, and the use of random-centered *vs.* object-centered queries. After a pilot study of 500 replications per experiment, we performed 10,000 replications per experiment.

5. Experimental Results

We now present the results[7] of the experiments just described. First, we summarize the effectiveness of the various alternative estimators. Second, we discuss some preliminary results on the use of heuristics for limiting the duration of index traversal. Finally, we demonstrate how the combination of traversal and sampling can greatly reduce the confidence interval widths.

[6] It is interesting to note that, for whatever reason, 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. Some argue that 20 equidepth buckets usually suffice and that 50 buckets should be adequate for skewed columns [IBM97b]; others make similar arguments with slightly higher values (40 and 100, respectively [INFO97b]); and some even place arbitrary (and relatively low) limits on the number of buckets (*e.g.*, 254 [ORAC97]).

[7] For completeness, we note that the experimental results were collected using libgist 1.0 on a ‘‘farm’’ of Dell PowerEdge 6100/200s running Solaris 2.6 (each configured with 4 200MHz Intel PentiumPro processors, 512MB of main memory, and one Seagate ST19171W Ultra Wide SCSI-3 disk drive storing operating system, swap and user data).

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.

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.

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. 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. The reason to limit the number of nodes is cost, both in terms of CPU processing (which is non-trivial) and I/Os (the exact number of which will depend on the buffering method in use). I/Os used during query optimization compete for disk arm time with actual query processing.

Tables 4 and 5 show two illustrative extracts from this snapshot. Full tables are contained in Appendix A. 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.)

The format of the tables is as follows. For each 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 in conjunction with conservative confidence intervals.[8] Note that the confidence interval widths are full widths, not half-widths.[9]

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 insertion-loaded index estimation implies that the traversal-based technique is best used with an access method that suits the query 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’’ [BELL61], and analogous degradations have been widely documented elsewhere in the statistics and computer science literature (a well-known treatment is contained in [SCOT92]). Third, random sampling performs quite poorly unless the selectivity is large (*i.e.*, $\gg 1\%$) because of the small sample sizes. Even accounting for this factor, the confidence interval widths are quite large — conservative confidence intervals are relatively weak and better results would be obtained if we could obtain enough samples to use better estimators.

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 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,

[8] 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 how many of its constituent intervals ‘‘stick out’’ beyond these limits.

[9] 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 $[0, N]$.

the correlation fractal dimension (D_2) estimator actually 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 reported in [BELU95], but this is to be expected given that the queries are highly 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 the hardest problems, namely, high dimensional, uniformly distributed data sets.

5.2. Effectiveness of Traversal Limit Heuristics

The previous subsection describes our experiments for a fixed traversal limit of 12 index nodes. This is a heuristic limit and we gave a heuristic argument for it. Recall that 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 exponential 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; see Figure 2.) 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 heuristic may be useful, we cannot recommend the naive tail-off heuristic for general use.

5.3. Effectiveness of Sampling with Traversal

We generally found that the simple estimator-switching rule of Section 4 was effective at determining when the traversal estimator was ‘‘stuck.’’ As an illlustration, Figure 3 shows the mean confidence interval width for one set of object-centered queries applied to the GTSPP (4D) data set. The mean confidence interval width for each of four separate estimation methods (index traversal only, traversal switching to conservative index sampling, conservative heap sampling, traversal switching to bootstrap index sampling) are plotted with their 95% confidence intervals (error bars). The estimator-switching rule moves to sampling fairly early, resulting in a final (p = 95%) conservative confidence interval that is half as wide as the traversal-only (p = 100%) confidence interval and very close to the sampling-only confidence interval.

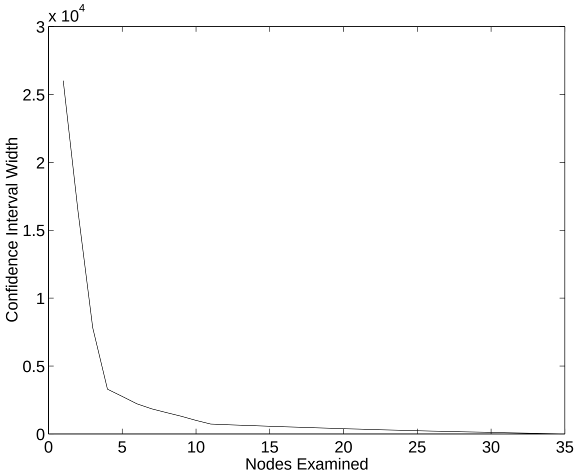


Figure 2. Slope change-points (Blob/3/o).

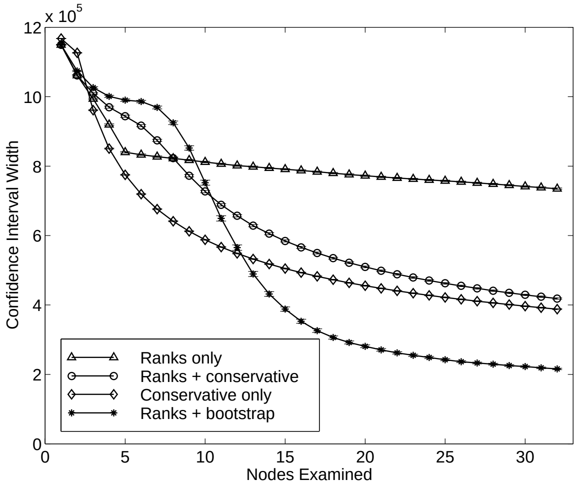


Figure 3. Mean c.i. widths (GTSPP/3/o).

The observant reader will have noted that the conservative sampling intervals are still quite large, and that sampling takes 30 node visits to achieve the 2:1 advantage. This is largely the result of using conservative bounds; reducing this interval width is an area of ongoing work. However, one must be very careful; because of the small sample sizes, experiments we conducted using CLT estimators produced high undercoverage rates, with coverage as low as 85% in some cases (recall that the nominal coverage is 95%!).[10]

We also computed non-parametric $\mathrm{BC_a}$ bootstrap confidence intervals. The bootstrap estimator produce coverage rates well above 95%. However, its benefit over conservative estimators is generally quite modest. Figure 3 happens to show the best case in our experiments. While the bootstrap estimator provides a 2:1 reduction in confidence interval width over the conservative estimator, the bootstrap confidence interval width is still over 15% of the total table size.

In summary, we have found that sampling techniques are useful in reducing confidence interval widths. Sampling forces examination of lower-level nodes in the cases (*i.e.*, low selectivity values) where index traversal is stuck in the upper regions of the index. However, much work remains in terms of improving the estimators themselves.

5.4. On the effects of pseudo-ranking

In Section 3, we noted that the (worst-case) effects of pseudo-ranking were easy to predict. We illustrate this in Table 6. Pseudo-ranked trees were insertion-loaded using the real data sets. The ranks were maintained using the bounds of [ANTO92] with parameters $A = 0.1$ and $Q = 0.3$, resulting in $\hat{N} \approx 1.15\ N$. Each pair of rows corresponds

[10] Note that the sampling undercoverage is actually somewhat *worse* than suggested by the numbers. In the not-infrequent case where index sampling *never* obtains a non-zero sample and the CLT estimator cannot produce a valid bound, we are actually reporting the deterministic confidence intervals that we had obtained from the preceding index traversal. Obviously, these never contribute to undercoverage.

Data / Scale / Query	Mean Rel. Error	Mean Abs. Error (%)	Max. Abs Error (%)	Mean c.i. Width (%)	Max. c.i. Width (%)
GNIS / 2 / o	0.0957	0.0199	0.2410	0.5123	5.7478
	0.1069	0.0223	0.2591	0.5828	6.3255
GNIS / 3 / o	0.0476	0.6159	2.7284	13.2880	33.5049
	0.0726	0.9396	3.5397	16.4154	39.1416
GTSPP / 2 / o	0.8553	0.1795	2.4018	49.3148	72.0985
	0.8597	0.1805	2.4039	54.7448	79.9434
GTSPP / 3 / o	0.3165	3.6874	12.3467	68.6537	81.2857
	0.3328	3.8773	12.6990	76.1930	90.1092
Blob / 2 / o	0.9869	0.8424	4.3740	34.8422	84.3703
	0.9876	0.8431	4.3770	38.6835	93.4858
Blob / 3 / o	0.7035	22.0057	45.2617	85.9696	98.1131
	0.7126	22.2907	45.7857	91.9598	100.0000

Table 6. Ranking *vs.* pseudo-ranking.

to data from object-centered queries over a ranked tree (taken from Table 4) and from a pseudo-ranked tree. Not only are the mean errors very similar, but the various maxima fall within the expected limits.

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 [MANN88] for background information about selectivity estimation; the references given here are generally incremental with respect to that survey.

6.1. Extensible estimation methods

The current state of the art is to provide black-box user hooks for selectivity estimation functions [INFO97a]. 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. Naturally, few useful selectivity estimators are written in practice.

6.2. Non-parametric statistics

Non-parametric density estimation encompasses a variety of techniques. Previous work in histogram methods and B-tree-related statistics resembles our work most closely. All of these approaches 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 reasonably ‘‘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 for the purpose of search.) Second, most are based on space-partitioning schemes; this results in summary data space requirements that are exponential in D.

Model-fitting techniques. Methods based on regression (*e.g.*, [CHEN94, GRAE87, SUN93], wavelets (*e.g.*, [MATI98]) and neural nets (*e.g.*, [BOUL97, LAKS98]) can be 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 (*e.g.*, [CHEN94]), the ability to perform dynamic updates of the summary data is generally limited.

Histograms. Now well-established (see [POOS97] for a recent survey), conventional histograms rely on space-partitioning schemes (which limits their applicability). When they can be applied, they consitute a very attractive option because of the many recent results on the generation of high-quality histograms, *e.g.*, [JAGA98]. None of this work deals with secondary memory data structures or hierarchical estimation.

Indexing main-memory multidimensional histograms with modified spatial access methods can speed up bucket retrieval [CHEN90, KAME85, MURA88]. SEDDS [GROS93] takes this idea further, using the spatial data structure to both construct and maintain the underlying histogram. Again, secondary memory and hierarchical estimation are not considered; 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 (see, *e.g.*, [ANTO93, WHAN94]). A somewhat more common use, found in the statistical access method literature, is to approximate distribution functions with an index (*cf.* [GHOS86, SRIV88] and [HINT91, MEIE97, WANG97]).

6.3. Tree traversal

An enormous literature exists on heuristic tree search in artificial intelligence. Much of this work deals with handling complexity (variable ordering, high levels of redundancy) that does not arise in our context. However, the priority-based algorithm of Section 3 could be superficially described as a type of best-first search.

With respect to database systems, the previous work in this area is discussed more thoroughly in Section 3 and Appendix B. The methods described in this paper makes the I/O budget explicit and then make the resulting imprecision explicit as well.

6.4. Sampling

Database sampling takes many forms and has many applications; see [OLKE95] and [BARB97, Sec. 9] for recent surveys. However, we have already described the most relevant index-assisted sampling literature in Section 3.

6.5. Tree condensation

Tree condensation always reduces the mean root-to-leaf path length and eliminates any dead-ends in the condensed region. However, static condensation schemes of the kinds considered in the literature (*e.g.*, condensing the top *k* levels, condensing every other level [KNUT75, ROSE93]) may not benefit a specific query at all.

Stratified tree-sampling techniques [CHEN92, GALL89] have a more dynamic flavor. The stratification may be defined in such a way as to create equivalence classes that vary in number depending on the data; however, it is sometimes difficult to limit the number of strata created, resulting in very expensive sampling plans.

Our combination of traversal followed by sampling has the effect of condensing the tree in a manner specific to the query, rather than according to a fixed structural or data-driven policy. We can therefore limit the condensation without altering the validity of our sampling plan.

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, 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 that addressed these problems.

Additional, smaller contributions include:

- 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.
- 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.
- We have illustrated that several of the proposed parametric estimators are unstable under reasonable conditions.

We have also shown that a combination of traversal and sampling can produce significant reductions in confidence interval width. Further reductions are an area of ongoing work.

Aside from the experimental and algorithmic improvements previously discussed, some immediate extensions include:

- Formal performance bounds. Can we do better than to provide examples of useful performance? Without additional assumptions or problem structure, there is no known ‘‘silver bullet’’ for multivariate density estimation that simultaneously provides useful bounds on space, time and precision. Furthermore, formal bounds of any kind are difficult in the framework of non-space-partitioning trees; a better understanding may arise from theoretical work on indexability [HELL97b].
- Sampling variations. Unlike simple random sampling, sampling from a ranked index has structure that might be exploitable. It would be interesting to formulate and investigate a sequential application of importance (non-equiprobable) sampling.
- Access path selection. The information obtained during single-table costing can help the optimizer decide which of two indices will be most useful for a given table. A conventional optimizer perceives two indices of the same type built over the same column as being identical. This is essentially true for B^+-trees but plainly untrue in general, as we can see if we consider the case of two R-trees which have been optimized for queries with inverse aspect ratios (*cf.* Proposition 1 of [HELL97b]). In the latter example, an estimator would only need to descend a short distance into each index to determine (on a heuristic basis, at least) that one is much better suited to a given query.
- Join selectivity estimation. Just as histograms on compatible domains can be ‘‘joined’’ to provide approximate joint frequency histograms, similar schemes are possible for index-assisted estimation. Applying our incremental techniques to this problem defines a middle ground between the previous work on costing index-assisted joins, which scans all [HUAN97] or none [THEO98] of the non-leaf index nodes.

Acknowledgements

The Blobworld data was obtained from Chad Carson and Megan Thomas. Experimental measurements reported here were obtained using resources funded by DARPA (F30602-95-C-0014), NSF (CDA 94-01156) and the California MICRO Program.

References

[ADEL97] B. Adelsman and R. Egan, “Using Indexes to Improve Query Performance,” Technical Document, IBM AS/400 Division, Rochester, MN, Sep. 1997. http://iws.as400.ibm.com/db2/db24ixp.htm.

[ANDE88] M.J. Anderson, R.L. Cole, W.S. Davidson, W.D. Lee, P.B. Passe, G.R. Ricard and L.W. Youngren, “Index Key Range Estimator,” U.S. Patent 4,774,657, IBM Corp., Armonk, NY, Sep. 1988.

[ANTO92] G. Antoshenkov, “Random Sampling from Pseudo-Ranked B^+ Trees,” *Proc. 18th Int’l Conf. on Very Large Data Bases*, Vancouver, BC, Canada, Aug. 1992, 375-382.

[ANTO93] G. Antoshenkov, “Dynamic Query Optimization in Rdb/VMS,” *Proc. 9th Int’l Conf. on Data Eng.*, Vienna, Austria, Apr. 1993, 538-547.

[AOKI98a] P.M. Aoki, “Generalizing ‘‘Search’’ in Generalized Search Trees,” *Proc. 14th Int’l Conf. on Data Eng.*, Orlando, FL, Feb. 1998, 380-389.

[AOKI98b] P.M. Aoki, “Algorithms for Index-Assisted Selectivity Estimation,” Tech. Rep. UCB//CSD-98-1021, Univ. of California, Berkeley, CA, Oct. 1998.

[BARB97] 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.

[BELL61] R.E. Bellman, *Adaptive Control Processes*, Princeton Univ. Press, Princeton, NJ, 1961. Cited in D. W. Scott, *Multivariate Density Estimation*, John Wiley & Sons, New York City, 1992..

[BELU95] A. Belussi and C. Faloutsos, “Estimating the Selectivity of Spatial Queries Using the ‘Correlation’ Fractal Dimension,” *Proc. 21st Int’l Conf. on Very Large Data Bases*, Zürich, Switzerland, Sep. 1995, 299-310.

[BLAS77] M.W. Blasgen and K.P. Eswaran, “Storage and Access in Relational Data Bases,” *IBM Sys. J. 16*, 4 (1977), 363-377.

[BOUL97] J. Boulos, Y. Viémont and K. Ono, “Analytical Models and Neural Networks for Query Cost Evaluation,” *Proc. 3rd Int’l Wksp. on Next Generation Inf. Technologies & Sys.*, Neve Ilan, Israel, July 1997, 138-149.

[CARE94] 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 ACM SIGMOD Int’l Conf. on Mgmt. of Data*, Minneapolis, MN, May 1994, 383-394.

[CARS97] C. Carson, S. Belongie, H. Greenspan and J. Malik, “Region-Based Image Querying,” *Proc. IEEE Wksp. on Content-Based Access of Image and Video Libraries*, San Juan, Puerto Rico, June 1997, 42-49.

[CHEN90] M.C. Chen, L. McNamee and N. Matloff, “Selectivity Estimation Using Homogeneity Measurement,” *Proc. 6th Int’l Conf. on Data Eng.*, Los Angeles, CA, Feb. 1990, 304-310.

[CHEN92] P.C. Chen, “Heuristic Sampling: A Method for Predicting the Performance of Tree Searching Programs,” *SIAM J. Comp. 21*, 2 (Apr. 1992), 295-315.

[CHEN94] C.M. Chen and N. Roussopoulos, “Adaptive Selectivity Estimation Using Query Feedback,” *Proc. 1994 ACM SIGMOD Int’l Conf. on Mgmt. of Data*, Minneapolis, MN, May 1994, 161-172.

[COCH77] W.G. Cochran, *Sampling Techniques (3rd Ed.)*, John Wiley & Sons, New York, 1977.

[COLE93] R.L. Cole, M.J. Anderson and R.J. Bestgen, “Query Processing in the IBM Application System 400,” *IEEE Data Eng. Bull. 16*, 4 (Dec. 1993), 19-28.

[COME79] D. Comer, “The Ubiquitous B-tree,” *Computing Surveys 11*, 2 (June 1979), 122-137.

[DICI96] T.J. DiCiccio and B. Efron, “Bootstrap Confidence Intervals,” *Stat. Sci. 11*, 3 (Aug. 1996), 189-228.

[FALO94] C. Faloutsos and I. Kamel, “Beyond Uniformity and Independence: Analysis of R-trees Using the Concept of Fractal Dimension,” *Proc. 13th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Sys.*, Minneapolis, MN, May 1994, 4-13.

[FARR94] 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.

[GALL89] P. Galle, “Branch and Sample: A Simple Strategy for Constraint Satisfaction,” *BIT 29*, 3 (1989), 395-408.

[GARC98] Y. J. García, M.A. López and S.T. Leutenegger, “On Optimal Node Splitting for R-trees,” *Proc. 24th Int’l Conf. on Very Large Data Bases*, New York City, Aug. 1998, 334-344.

[GHOS86] S. Ghosh, “SIAM: Statistics Information Access Method,” in *Proc. 3rd Int’l Wksp. on Stat. & Sci. Database Mgmt.* (Luxembourg, July 1986), R. Cubitt, B. Cooper and G. Özsoyoğlu (eds.), EUROSTAT, Luxembourg, 1986, 286-293.

[GRAE87] G. Graefe, “Selectivity Estimation Using Moments and Density Functions,” Tech. Rep. CS/E 87-012, Oregon Graduate Center, Beaverton, OR, Nov. 1987.

[GROS93] W.I. Grosky, J. Sun and F. Fotouhi, “Dynamic Selectivity Estimation for Multidimensional Queries,” in *Foundations of Data Organization and Algorithms* (Proc. 4th Int’l Conf., Chicago, IL, Oct. 1993), D.B. Lomet (ed.), Springer Verlag, LNCS Vol. 730, Berlin, 1993, 231-246.

[GUTT84] A. Guttman, “R-trees: A Dynamic Index Structure for Spatial Searching,” *Proc. 1984 ACM SIGMOD Int’l Conf. on Mgmt. of Data*, Boston, MA, June 1984, 47-57.

[HAAS97] P.J. Haas, “Large-Sample and Deterministic Confidence Intervals for Online Aggregation,” *Proc. 9th Int’l Conf. on Stat. & Sci. Database Mgmt.*, Olympia, WA, Aug. 1997, 51-62.

[HAMI94] 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.

[HELL95] J.M. Hellerstein, J.F. Naughton and A. Pfeffer, “Generalized Search Trees for Database Systems,” *Proc. 21st Int’l Conf. on Very Large Data Bases*, Zürich, Switzerland, Sep. 1995, 562-573.

[HELL97a] J.M. Hellerstein, P.J. Haas and H. Wang, “Online Aggregation,” *Proc. 1997 ACM SIGMOD Int’l Conf. on Mgmt. of Data*, Tucson, AZ, May 1997, 171-182.

[HELL97b] J.M. Hellerstein, E. Koutsoupias and C.H. Papadimitriou, “On the Analysis of Indexing Schemes,” *Proc. 16th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Sys.*, Tucson, AZ, May 1997, 249-256.

[HINT91] H. Hinterberger, “Visualizing Patterns in Multidimensional Spaces: Density-Displays to Trade Detail for Speed,” in *Statistical and Scientific Databases*, Z. Michalewicz (ed.), Ellis Horwood, New York, 1991, 83-108.

[HOEF63] W. Hoeffding, “Probability Inequalities for Sums of Bounded Random Variables,” *J. Am. Stat. Assoc. 58*, 301 (Mar. 1963), 13-30.

[HUAN97] Y.-W. Huang, N. Jing and E. Rundensteiner, “A Cost Model for Estimating the Performance of Spatial Joins Using R-trees,” *Proc. 9th Int’l Conf. on Stat. & Sci. Database Mgmt.*, Olympia, WA, Aug. 1997, 30-38.

[IBM97a] “Query Optimizer Seize,” Technical Document 6511134, IBM AS/400 Division, Rochester, MN, Aug. 1997. http://as400service.rochester.ibm.com/.

[IBM97b] “IBM DB2 Universal Database Administration Guide, Version 5,” Document No. S10J-8157-00, IBM Corp., North York, Ontario, Canada, 1997.

[IBM98] “DB2 Performance Tuning on VSE and VM,” Document No. SG24-5146-00, IBM Corp., Böblingen, Germany, May 1998.

[ILLU95] “Illustra User’s Guide, Server Release 3.2,” Part No. DBMS-00-42-UG, Illustra Information Technologies, Inc., Oakland, CA, Oct. 1995.

[INFO97a] “Guide to the Virtual-Table Interface, Version 9.01,” Part No. 000-3692, Informix Corp., Menlo Park, CA, Jan. 1997.

[INFO97b] “Informix Universal Server Performance Guide, Version 9.01,” Part No. 000-3699, Informix Corp., Menlo Park, CA, Jan. 1997.

[JAGA90] H.V. Jagadish, “Linear Clustering of Objects with Multiple Attributes,” *Proc. 1990 ACM SIGMOD Int’l Conf. on Mgmt. of Data*, Atlantic City, NJ, May 1990, 332-342.

[JAGA98] H.V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K.C. Sevcik and T. Suel, “Optimal Histograms with Quality Guarantees,” *Proc. 24th Int’l Conf. on Very Large Data Bases*, New York City, Aug. 1998, 275-286 .

[JAIN91] R. Jain, *The Art of Computer Systems Performance Analysis*, John Wiley & Sons, New York, 1991.

[KAME85] N. Kamel and R. King, “A Model of Data Distribution Based on Texture Analysis,” *Proc. 1985 ACM SIGMOD Int’l Conf. on Mgmt. of Data*, Austin, TX, May 1985, 319-325.

[KAME93] I. Kamel and C. Faloutsos, “On Packing R-trees,” *Proc. 2nd Int’l Conf. on Inf. & Knowledge Mgmt.*, Arlington, VA, Nov. 1993, 490-499.

[KIM96] S.-W. Kim, W.-K. Whang and K.-Y. Whang, “Analyzing Errors in Selectivity Estimation Using the Multilevel Grid File” [Korean, English abstract], *J. Korean Inst. Telematics & Electronics 33B*, 9 (Sep. 1996), 24-36.

[KIRK96] J. Kirkwood, *Sybase SQL Server 11: An Administrator’s Guide*, Thomson Comp. Press, Boston, 1996.

[KNUT73] D.E. Knuth, *The Art of Computer Programming, Vol. III: Sorting and Searching*, Addison Wesley, Reading, MA, 1973.

[KNUT75] D.E. Knuth, “Estimating the Efficiency of Backtrack Programs,” *Math. Comput. 29*, 129 (Jan. 1975), 121-136.

[LAKS98] S. Lakshmi and S. Zhou, “Selectivity Estimation in Extensible Databases - A Neural Network Approach,” *Proc. 24th Int’l Conf. on Very Large Data Bases*, New York City, Aug. 1998, 623-627.

[LEUT97] S.T. Leutenegger, M.A. López and J.M. Edgington, “STR: A Simple and Efficient Algorithm for R-tree Packing,” *Proc. 13th Int’l Conf. on Data Eng.*, Birmingham, England, Apr. 1997, 497-506.

[MANN88] M.V. Mannino, P. Chu and T. Sager, “Statistical Profile Estimation in Database Systems,” *Computing Surveys 20*, 3 (Sep. 1988), 191-221.

[MATI98] Y. Matias, J.S. Vitter and M. Wang, “Wavelet-Based Histograms for Selectivity Estimation,” *Proc. 1998 ACM SIGMOD Int’l Conf. on Mgmt. of Data*, Seattle, WA, May 1998, 448-459.

[MEIE97] K.A. Meier, “Data Abstraction Through Density Estimation by Storage Management,” *Proc. 9th Int’l Conf. on Stat. & Sci. Database Mgmt.*, Olympia, WA, Aug. 1997, 39-50.

[MURA88] M. Muralikrishna and D.J. DeWitt, “Equi-depth Histograms for Estimating Selectivity Factors for Multi-Dimensional Queries,” *Proc. 1988 ACM SIGMOD Int’l Conf. on Mgmt. of Data*, Chicago, IL, June 1988, 28-36.

[OLKE89] F. Olken and D. Rotem, “Random Sampling from B^+ Trees,” *Proc. 15th Int’l Conf. on Very Large Data Bases*, Amsterdam, the Netherlands, Aug. 1989, 269-277.

[OLKE95] F. Olken and D. Rotem, “Random Sampling from Databases: A Survey,” *Statistics & Computing 5*, 1 (Mar. 1995), 25-42.

[ORAC97] “Oracle8 Server SQL Reference, Release 8.0,” Part No. A54647-01, Oracle Corp., Redwood Shores, CA, June 1997.

[PAGE93] B.-U. Pagel, H.-W. Six, H. Toben and P. Widmayer, “Toward an Analysis of Range Query Performance in Spatial Data Structures,” *Proc. 12th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Sys.*, Washington, DC, May 1993, 214-221.

[POOS97] V. Poosala, *Histogram-Based Estimation Techniques in Database Systems*, Ph.D. dissertation, Univ. of Wisconsin, Madison, WI, July 1997. UMI No. 9716074.

[ROSE93] P.R. Rosenbaum, “Sampling the Leaves of a Tree with Equal Probabilities,” *J. Am. Stat. Assoc. 88*, 424 (Dec. 1993), 1455-1457.

[SCOT92] D.W. Scott, *Multivariate Density Estimation: Theory, Practice, and Visualization*, John Wiley & Sons, New York, 1992.

[SMIT96] I. Smith, “Oracle Rdb: What’s New,” in *DECUS Spring ’96* (St. Louis, MO), DECUS, Littleton, MA, June 1996, IM-016.

[SRIV88] J. Srivastava and V.Y. Lum, “A Tree Based Access Method (TBSAM) for Fast Processing of Aggregate Queries,” *Proc. 4th Int’l Conf. on Data Eng.*, Los Angeles, CA, Feb. 1988, 504-510.

[STON93] M. Stonebraker, J. Frew, K. Gardels and J. Meredith, “The Sequoia 2000 Storage Benchmark,” *Proc. 1993 ACM SIGMOD Int’l Conf. on Mgmt. of Data*, Washington, DC, May 1993, 2-11.

[SUN93] W. Sun, Y. Ling, N. Rishe and Y. Deng, “An Instant and Accurate Size Estimation Method for Joins and Selections in a Retrieval-Intensive Environment,” *Proc. 1993 ACM SIGMOD Int’l Conf. on Mgmt. of Data*, Washington, DC, May 1993, 79-88.

[THEO96] Y. Theodoridis and T. Sellis, “A Model for the Prediction of R-tree Performance,” *Proc. 15th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Sys.*, Montreal, Québec, Canada, June 1996, 161-171.

[THEO98] Y. Theodoridis, E. Stefanakis and T. Sellis, “Cost Models for Join Queries in Spatial Databases,” *Proc. 14th Int’l Conf. on Data Eng.*, Orlando, FL, Feb. 1998, 476-483.

[USGS95] U. S. Geological Survey, “Geographic Names Information System,” Data Users Guide 6 (4th printing, revised), U. S. Department of the Interior, Reston, VA, 1995.

[WANG97] W. Wang, J. Yang and R. Muntz, “STING: A Statistical Information Grid Approach to Spatial Data Mining,” *Proc. 23rd Int’l Conf. on Very Large Data Bases*, Athens, Greece, Aug. 1997, 186-195.

[WHAN94] K.-Y. Whang, S.-W. Kim and G. Wiederhold, “Dynamic Maintenance of Data Distribution for Selectivity Estimation,” *VLDB J. 3*, 1 (Jan. 1994), 29-51.

[WONG80] C.K. Wong and M.C. Easton, “An Efficient Method for Weighted Sampling Without Replacement,” *SIAM J. Comp. 9*, 1 (Feb. 1980), 111-113.

Appendix A: Experimental Results

This appendix summarizes our experiments comparing traversal-based estimation, random sampling, and various parametric estimators. The formatting and notation are identical to that of the tables in Section 5.

Some of the results of Tables A.2 and A.4 have been excluded. These tables pertain to the uniform-centered queries. With the high dimensionality data, the mean selectivity was so close to zero as to be meaningless.

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 / 1 / o	0.0039	0.0001	i	ranks	0.1806	0.0007	0.0815	0.0278	2.3226
			b	ranks	0.0215	0.0001	0.0473	0.0004	0.1964
				sample	1.7440	0.0068	1.4770	35.3336	36.8142
				D_2	0.6990	0.0027	0.1851		
GNIS / 2 / o	0.2085	0.0014	i	ranks	0.0957	0.0199	0.2410	0.5123	5.7478
			b	ranks	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	ranks	0.0476	0.6159	2.7284	13.2880	33.5049
			b	ranks	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 / 1 / o	0.0121	0.0000	i	ranks	0.9981	0.0121	0.1314	31.3904	63.6347
			b	ranks	0.7084	0.0086	0.1258	0.0861	1.4444
				sample	0.9722	0.0118	0.4004	35.3424	35.8510
				D_2	1.2900	0.0156	0.1216		
GTSPP / 2 / o	0.2099	0.0002	i	ranks	0.8553	0.1795	2.4018	49.3148	72.0985
			b	ranks	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	ranks	0.3165	3.6874	12.3467	68.6537	81.2857
			b	ranks	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 / 1 / o	0.0414	0.0003	i	ranks	0.8794	0.0364	1.4988	13.0217	57.8225
			b	ranks	0.0495	0.0020	0.2782	0.0148	5.4648
				sample	0.8682	0.0360	3.4129	35.3714	40.1689
				D_2	5.4220	0.2246	1.2790		
Blob / 2 / o	0.8536	0.0004	i	ranks	0.9869	0.8424	4.3740	34.8422	84.3703
			b	ranks	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	ranks	0.7035	22.0057	45.2617	85.9696	98.1131
			b	ranks	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 A.1. Real data, object-centered queries — 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 (%)
GNIS / 1 / u	0.0007	0.0000	i	ranks	0.1110	0.0001	0.0190	0.0084	2.3552
			b	ranks	0.0000	0.0000	0.0001	0.0000	0.0098
				sample	1.9881	0.0013	0.3392	35.3309	35.6726
				uniform	1.0012	0.0007	0.0462		
				D_0	1.0119	0.0007	0.0461		
			i	density	1.2071	0.0008	0.0456		
			b	density	1.0694	0.0007	0.0459		
GNIS / 2 / u	0.0674	0.0009	i	ranks	0.1093	0.0074	0.2307	0.1968	5.1020
			b	ranks	0.0201	0.0014	0.1184	0.0368	0.5893
				sample	1.4119	0.0951	3.7289	35.4001	39.2115
				uniform	0.9798	0.0660	1.0920		
				D_0	0.9783	0.0659	1.0911		
			i	density	0.9741	0.0656	1.0885		
			b	density	0.9762	0.0658	1.0899		
GNIS / 3 / u	6.4113	0.0560	i	ranks	0.0502	0.3216	2.7517	6.8648	33.4458
			b	ranks	0.0639	0.4096	4.2398	5.6194	25.5467
				sample	0.6317	4.0499	41.6211	41.7154	70.6604
				uniform	0.8864	5.6832	26.4125		
				D_0	0.8856	5.6777	26.4038		
			i	density	0.8832	5.6624	26.3799		
			b	density	0.8844	5.6700	26.3918		

Table A.2. Real data, random-centered queries — 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 (%)
Uni2 / 1 / o	0.0007	0.0000	i	ranks	0.2741	0.0002	0.0013	3.6643	13.5989
			b	ranks	0.0000	0.0000	0.0000	0.0000	0.0000
				sample	2.0897	0.0015	0.0568	35.3310	35.3873
				D_2	0.3831	0.0003	0.0011		
Uni2 / 2 / o	0.0652	0.0001	i	ranks	0.1553	0.0101	0.0633	4.9770	16.0626
			b	ranks	0.0055	0.0004	0.0023	0.0228	0.0589
				sample	0.7329	0.0478	0.3897	35.3952	35.7868
				D_2	0.3948	0.0258	0.0352		
Uni2 / 3 / o	5.7602	0.0153	i	ranks	0.0768	0.4424	1.7628	20.3640	37.0192
			b	ranks	0.0326	0.1875	0.6832	5.2875	8.5947
				sample	0.0764	0.4403	2.5056	41.0828	44.4626
				D_2	0.3358	1.9343	2.7139		
Uni4 / 1 / o	0.0001	0.0000	i	ranks	0.9940	0.0001	0.0002	39.1866	78.3194
			b	ranks	0.0007	0.0000	0.0001	0.0000	0.0252
				sample	1.9056	0.0002	0.0867	35.3303	35.4170
				D_2	0.0051	0.0000	0.0001		
Uni4 / 2 / o	0.0020	0.0000	i	ranks	0.3019	0.0006	0.0033	45.0454	82.9009
			b	ranks	0.0620	0.0001	0.0016	0.0275	0.2266
				sample	1.9954	0.0040	0.1717	35.3323	35.5038
				D_2	0.9572	0.0019	0.0039		
Uni4 / 3 / o	4.2810	0.0230	i	ranks	0.1739	0.7443	4.0287	76.8423	96.1341
			b	ranks	0.0265	0.1134	0.5918	16.4368	32.8581
				sample	0.1081	0.4627	2.7488	39.6068	45.6600
				D_2	1.0000	4.2809	8.2977		
Uni20 / 1 / o	0.0038	0.0000	i	ranks	0.9996	0.0038	0.0038	67.9227	90.7075
			b	ranks	0.0165	0.0001	0.0038	0.0330	8.5892
				sample	1.9921	0.0077	0.2650	35.3341	35.5990
				D_2	0.0000	0.0000	0.0000		
Uni20 / 2 / o	0.0038	0.0000	i	ranks	1.0000	0.0038	0.0038	88.6628	92.6213
			b	ranks	0.2815	0.0011	0.0038	1.0388	61.6002
				sample	2.0872	0.0080	0.2650	35.3345	35.5990
				D_2	0.0000	0.0000	0.0000		
Uni20 / 3 / o	0.0107	0.0001	i	ranks	0.4202	0.0045	0.0696	96.3064	99.0200
			b	ranks	0.4229	0.0045	0.0501	20.9365	100.0000
				sample	1.8263	0.0195	0.5299	35.3403	35.8678
				D_2	0.6408	0.0069	0.1307		

Table A.3. Uniform data, object-centered queries — 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 (%)
Uni2 / 1 / u	0.0007	0.0000	i	ranks	0.2983	0.0002	0.0012	3.6938	13.5989
			b	ranks	0.0000	0.0000	0.0000	0.0000	0.0000
				sample	2.0461	0.0013	0.1137	35.3309	35.4443
				uniform	1.0000	0.0007	0.0016		
				D_0	0.8066	0.0005	0.0015		
			i	density	20.5691	0.0136	0.0142		
			b	density	1635.8551	1.0788	1.0794		
Uni2 / 2 / u	0.0651	0.0001	i	ranks	0.1551	0.0101	0.0654	4.9977	16.0513
			b	ranks	0.0055	0.0004	0.0028	0.0228	0.0589
				sample	0.7353	0.0479	0.3335	35.3957	35.7297
				uniform	1.0000	0.0651	0.0745		
				D_0	0.9886	0.0644	0.0738		
			i	density	0.3703	0.0241	0.0334		
			b	density	0.9965	0.0649	0.0743		
Uni2 / 3 / u	5.7692	0.0152	i	ranks	0.0773	0.4462	1.7876	20.3160	36.8384
			b	ranks	0.0330	0.1902	0.7171	5.3158	8.5947
				sample	0.0767	0.4428	2.2289	41.0942	44.1201
				uniform	1.0000	5.7691	6.6631		
				D_0	0.9975	5.7545	6.6485		
			i	density	0.9317	5.3750	6.2690		
			b	density	0.9999	5.7686	6.6625		
Uni4 / 1 / u	0.0000	0.0000	i	ranks	2.0617	0.0000	0.0001	39.3343	79.1562
			b	ranks	0.0002	0.0000	0.0000	0.0000	0.0336
				sample	1.0000	0.0000	0.0001	35.3302	35.3302
				uniform	1.0106	0.0000	0.0001		
				D_0	2015.7055	0.0010	0.0010		
			i	density	111399.7636	0.0534	0.0534		
			b	density	6272.0410	0.0030	0.0030		
Uni4 / 2 / u	0.0019	0.0000	i	ranks	0.3139	0.0006	0.0035	44.9318	82.7869
			b	ranks	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	ranks	0.1729	0.7469	4.0015	77.0006	95.9137
			b	ranks	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 A.4. Uniform data, uniform-centered queries — 12 nodes.

Appendix B: Traversal

This appendix provides a somewhat more systematic view of the index traversal literature than Section 2. Various strategies for estimating the cardinality of a subtree, estimating the number of records matching a predicate and traversing the tree have been proposed. We discuss each in turn.

First, we need some way of estimating how many records are contained in a given subtree. Cardinality estimation methods may be:

- *Fanout-based*. Given a fanout estimate f, a subtree rooted at a node that is l levels above the leaf level may be estimated to contain f^{l+1} records [ANDE88, ANTO93, OLKE89].
- *Rank-based*. Say we store the cardinality of every subtree (or an approximation) in its root. For historical reasons, trees maintaining such cardinality counts are known as *ranked trees* [KNUT73]; trees with approximate counts (albeit with upper and lower bounds) are called *pseudo-ranked* [ANTO92], and conventional trees that do not maintain extra information are called *unranked*. Given such ranks, or approximate ranks, cardinality estimation is trivial [ANTO92, WHAN94]. Note that pseudo-ranking subsumes both ranked and unranked trees as special cases.

These options essentially divide the design space into methods that do not use local information (*i.e.*, ranks) and those that do.

Second, because the query predicate may overlap but not contain the node entry predicate, we need some way to guess how many records in a subtree actually match the query predicate. Partial-match estimation methods include:

- *Fixed proportion*. We can guess that some arbitrary fraction (*e.g.*, all or half) of the records match the query [ANDE88, ANTO93, MURA88].
- *Overlap-based*. We can measure the overlap between the node entry predicate and the query predicate and apportion the records accordingly [MURA88, WHAN94].

Again, these methods divide the design space into two: fixed and variable overlap estimates.

Third, we need to understand how to traverse the tree to best apply the previous two mechanisms. The most precise information is plainly at the lowest point(s) reached by a traversal. Hence, the methods described below descend to a level at which estimation (cardinality and/or partial match) is performed. Traversal methods proposed in the literature include:

- *Fixed level*. Descends the tree top-down to a tree level chosen *a priori*, following all links whose predicate matches the query. First proposed for the root of a tree [BLAS77] but can be generalized to any other level [ANDE88, COLE93, KIM96, WHAN94].
- *Split level*. Descends the tree top-down until more than one link matches the query (*i.e.*, until one reaches the lowest single node that ‘‘covers’’ the query — this may be the root). This was implemented by Rdb/VMS [ANTO92, ANTO93].
- *Depth-first*. TBSAM [SRIV88] does depth-first search on all partial matches to compute exact cardinality results.

Note that, unlike the methods discussed in the preceding two paragraphs, these methods do not taxonomize the traversal design space.

It is important to note at this point that each of the three different sets of strategies result in some imprecision. That is, applying them results in an estimate, but the estimate is really a value within an interval of possible values. For example, even though we may guess that half of a subtree’s records match when its predicate overlaps with the query, the actual answer is that anywhere from zero to all of its records may match.

Figure B.1[11] demonstrates how two commercial systems, IBM DB2/400 [ADEL97] and Oracle Rdb 7.0 [SMIT96], perform index-assisted estimation on B^+-trees.[12] DB2/400 performs fanout-based cardinality estimation, assumes 100% overlap on partial matches and descends to a fixed level (the level above the leaf nodes). Oracle Rdb uses pseudo-ranked cardinality estimation, assumes 50% overlap on partial matches and performs split-level descent. Figure B.2 illustrates how TBSAM works in the same situation. (In practice, the traversal algorithm of Section 3 works something like this.)

[11] Reproduced from [AOKI98a].

[12] In fact, [ANDE88] describes the use of binary radix trees, which are not height-balanced. The discussion here therefore extends the ideas of [ANDE88]. For example, we ignore the ‘‘pilot probes,’’ which only serve to estimate the radix tree’s height.

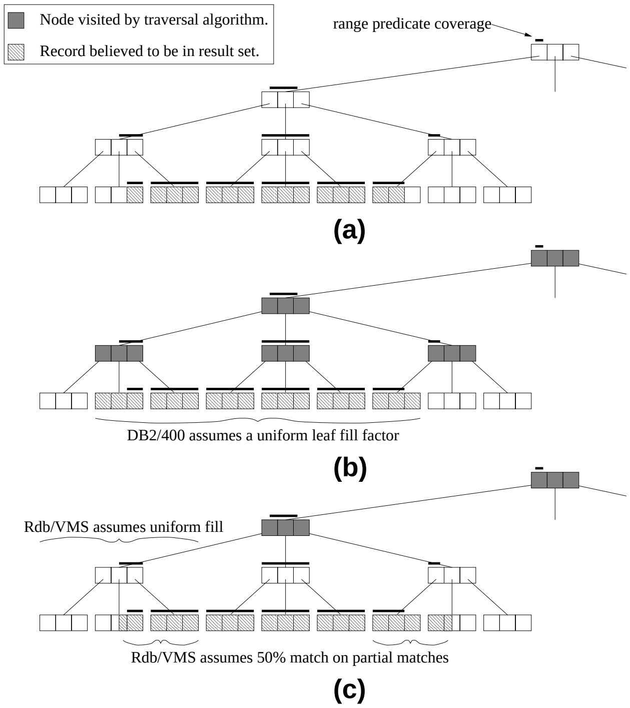

Node visited by traversal algorithm.
Record believed to be in result set.
range predicate coverage
(a)
DB2/400 assumes a uniform leaf fill factor
(b)
Rdb/VMS assumes uniform fill
Rdb/VMS assumes 50% match on partial matches
(c)

Figure B.1. B$^+$-tree descent strategies:
(a) The actual query predicate coverage.
(b) Descent to level above leaves in DB2/400.
(c) Descent to “split level” in Rdb/VMS.

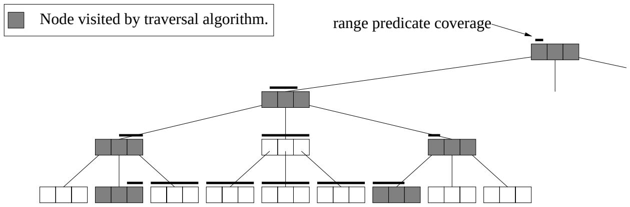


Figure B.2. TBSAM `COUNT` estimation strategy.

Appendix C: Tree traversal, ranking schemes and estimation accuracy

Throughout this paper, we posit that a pseudo-ranked tree satisfies certain conditions that allow us to reason about the cardinality bounds. In this appendix, we (1) define these conditions, (2) show that these conditions are sufficient to guarantee that our deterministic bounds never suffer while we probe the tree, and (3) give expressions for the maximum and minimum amounts by which our deterministic bounds tighten given a specific type of pseudo-ranking.

Recall the following two definitions and notation from Section 2.1:

Definition C.1 [ANTO92]: *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 $e.\,ptr$.*

Definition C.2 [ANTO92]: *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.*

All summations in this appendix will be over $j \in child(i)$.

1. Suboptimality of traversal strategies

A useful consequence of the nested bound condition is that we cannot pick a traversal strategy that ‘‘loses ground.’’ That is, if our ranking scheme satisfies the nested bound condition, the deterministic bounds on our cardinality estimate never grow if we descend into a new node $child(i)$. Denote the uncertainty of the node entry i by u and the cumulative uncertainty of $j \in child(i)$ by u'.

Observation C.1: *Descending a pointer from i to $child(i)$ never increases uncertainty (i.e., $u \ge u'$) if the index satisfies the nested bound condition.*

Proof: If we follow a pointer, the query predicate q and the predicate of i must either be a full match (q contains i's predicate) or a partial match (q overlaps but does not contain i's predicate). Each of these cases turns out to have different uncertainty.

Case 1 (full match): The number of records that match q falls somewhere within the range of possible subtree cardinalities, $[c^-, c^+]$. Hence, $u = c_i^+ - c_i^-$ and $u' = \sum(c_j^+ - c_j^-)$. By the nested bounds condition,

$$u = c_i^+ - c_i^- \;\ge\; \left(\sum c_j^+\right) - c_i^- \;\ge\; \left(\sum c_j^+\right) - \left(\sum c_j^-\right) \;=\; \sum(c_j^+ - c_j^-) = u'$$

Case 2 (partial match): The number of records under i that match p might be anywhere on $[0, c_i^+]$, so $u = c_i^+$. Similarly, the cumulative uncertainty of the child node's entries is $u' = \sum c_j^+$ (all of the entries can be partial matches in, say, an R-tree). $u \ge u'$ follows immediately from the nested bounds condition. □

Note that Observation C.1 does not imply that the estimate always becomes more accurate. Even if the deterministic bounds shrink (which they are not guaranteed to do), the estimate itself may move away from the true value.

2. Alternative traversal strategies

The reason why we want to reduce the ‘‘uncertainty’’ is that any overestimation of the total cardinality translates directly into loss of sampling efficiency. Two kinds of strategies suggest themselves. First, we can choose node entries that give the best worst-case uncertainty reduction. This is a pessimistic, minimax-style strategy. Second, we can choose node entries that give the best best-case uncertainty reduction. This is an optimistic, maximin-style strategy.

Consider how these strategies work under the most general conditions, *i.e.*, we know only that the nested bound condition applies. Without any other assumptions, the worst-case uncertainty reduction is zero. In this case, then, the worst-case strategy gives us no information. The best-case uncertainty reduction is 100% (reduction to zero).

Therefore, this strategy amounts to choosing the node entry with the greatest uncertainty.

Other strategies are possible if we constrain the kind of bounds we use. Let us assume that $c_i^+ = c_i^0 \cdot \varepsilon_h$ and $c_i^- = c_i^0 / \varepsilon_h$, where h is the height in the tree and ε_h is some function that is non-decreasing in h. When descending from i to $child(i)$, we can be sure that the uncertainty decreases by at least

$$u - u' = \left(c_i^+ - c_i^-\right) - \left(\sum c_j^+ - \sum c_j^-\right) = \left(c_i^+ - \left(\frac{c_i^+}{{\varepsilon_h}^2}\right)\right) - \left(\sum c_j^+ - \left(\sum \frac{c_j^+}{{\varepsilon_{h+1}}^2}\right)\right) = \left(c_i^+ - \left(\frac{c_i^+}{{\varepsilon_h}^2}\right)\right) - \left(c_i^+ - \left(\frac{c_i^+}{{\varepsilon_{h+1}}^2}\right)\right)$$

$$= c_i^+\left(\frac{1}{{\varepsilon_{h+1}}^2} - \frac{1}{{\varepsilon_h}^2}\right)$$

since the difference is minimized when $c_i^+ = \sum c_j^+$. (Of course, this difference may be very small, or even zero.) By a similar argument, a full match reduces the uncertainty by at most

$$u - u' = c_i^-\left({\varepsilon_h}^2 - {\varepsilon_{h+1}}^2\right)$$

A partial match may reduce the uncertainty from c_i^+ to 0 (*i.e.*, when no entries $j \in child(i)$ have a predicate that overlaps the query predicate q). These bounds are summarized in Table C.1; they represent the worst-case and best-case "payoff" for following a node entry pointer. If our nested bounds happen to be these type of ε bounds, we can use either of these payoff metrics (instead of the largest uncertainty metric) when choosing which node entry pointer to follow.

	Minimum decrease	Maximum decrease
Full match	$c_i^+\left(\frac{1}{{\varepsilon_{h+1}}^2} - \frac{1}{{\varepsilon_h}^2}\right)$	$c_i^-\left({\varepsilon_h}^2 - {\varepsilon_{h+1}}^2\right)$
Partial match		c_i^+

Table C.1. Uncertainty reduction with ε bounds.

Appendix D: A/R sampling from any kind of ranked tree

Figure D.1 shows the basic acceptance/rejection (A/R) sampling algorithm for trees that conform to the nested bound condition. The purpose of the algorithm is to select one record out of N

- at random,
- with equal probability, and
- without precise information about N.

The algorithm uses some notation beyond what was defined in Table 1: $|node|$ indicates the fanout (outdegree) of *node*, and $node[i]$ indicates the ith node entry (index record) of *node*.

```
 1.  Set $node = root$
 2.  Set $\hat{c} = \sum_{i=1}^{|node|} c^{+}_{node[i]}$                    // initial estimate of tree size
 3.      Pick random $x \in [1, \hat{c}]$
 4.      If $x > \sum_{i=1}^{|node|} c^{+}_{node[i]}$, goto 1                // REJECT (start a new probe)
 5.      Find smallest $j \in [1, |node|]$ s.t. $\sum_{i=1}^{j} c^{+}_{node[i]} \geq x$
 6.      If $node[j].ptr$ = NULL, goto 10                                      // this is a record
 7.      Set $\hat{c} = c^{+}_{node[j]}$                                         // estimate of new subtree size
 8.      Set $node = node[j].ptr$                                              // descend to new subtree
 9.      Goto 3
10.  Return $node[j]$                                                          // ACCEPT
```

Figure D.1. Algorithm A/R [ANTO92].

Each paper on acceptance/rejection sampling from trees gives an *ad hoc* argument for the equiprobability of the A/R algorithm. Antoshenkov [ANTO92] makes a correct but informal argument that a balanced pseudo-ranked tree can be conceptually transformed into an equivalent balanced ranked tree with ‘‘phantom’’ edges. The fact that balanced unranked trees could be so transformed is pointed out in [OLKE89, p. 273]; Olken and Rotem, as well as Rosenbaum [ROSE93], give equiprobability proofs specific to their ranking schemes that do not depend on this fact. However, we can also make a more satisfying direct proof for all tree topologies (balanced or unbalanced) and all ranking schemes that support nested cardinality bounds.

The general framework of Theorem D.1 is taken from [ROSE93]. We have replaced his algorithm-specific version of Lemma D.1.1 with a general version.

Theorem D.1: *Given a tree of cardinality N, a series of A/R descents that halt after returning one record will return a specific record r with probability* $\mathbf{Pr}[accept\ r] = 1/N$.

We will assume that the early abort A/R algorithm given above is in use, but the results apply to the lazy abort algorithm as well.

Lemma D.1.1: *Each record will be returned with equal probability $1/\hat{N}$ (where $\hat{N} = \sum_{e \in root} c^{+}_{e}$) during a single A/R descent.*

Let h be the height (maximum depth) of the tree. Consider a path from the root to some index record, *i.e.*, a path representing a single successful descent. This path consists of index entries $e_0, \ldots, e_l$, $0 \leq l \leq h$, each drawn from a corresponding index node $n_0, \ldots, n_l$. Entry e_k contains an upper bound estimate, $c^{+}_{e_k}$, of the subtree rooted at node n_{k+1}; define $c^{+}_{e_l} = 1$ (e_l is a leaf record and therefore has no children). The A/R algorithm selects entry e_k with probability $c^{+}_{e_k}/c^{+}_{e_{k-1}}$, so the probability of accepting a given record r is $\mathbf{Pr}[e_l = r] = \frac{c^{+}_{e_0}}{\hat{N}} \prod_{k=1}^{l} \frac{c^{+}_{e_k}}{c^{+}_{e_{k-1}}}$. This expression telescopes, resulting in $\mathbf{Pr}[e_l = r] = 1/\hat{N}$ for all r. □

Lemma D.1.1 demonstrates that the equiprobability is inherent in the A/R procedure rather than a result of a particular ranking scheme or tree topology. It also follows from the lemma that the probability of accepting some record during a single descent is $A = N/\hat{N}$ and the probability of rejection is $R = 1 - A = (\hat{N} - N)/\hat{N}$.

Lemma D.1.2 [ROSE93]: *Each record will be returned with equal probability by a series of descents that halts after returning one record.*

We now wish to compute the probability of eventually accepting record r after some number of descents. Consider the possible events that can occur from the standpoint of a single descent:

Event	Prob.	
accept ($= r$) now	$1/\hat{N}$	$= A$
accept ($\neq r$) now	$(N - 1)/\hat{N}$	
reject; accept ($= r$) later	Ra_r	$= R$
reject; accept ($\neq r$) later	$R(1 - a_r)$	

Only the first and third events result in record r being returned, so $\mathbf{Pr}[\textit{accept } r] = 1/\hat{N} + R \cdot \mathbf{Pr}[\textit{accept } r]$. (The remaining event probabilities correspond to acceptance of a different record, which is not of interest here.) Solving for $\mathbf{Pr}[\textit{accept } r]$, we obtain $\mathbf{Pr}[\textit{accept } r] = 1/(\hat{N}(1 - R))$, which is equal for all r. □

Since $R = 1 - A$, it follows that $\mathbf{Pr}[\textit{accept } r] = 1/(\hat{N}A) = 1/N$. □