# Random Sampling from $B^+$ trees *

Frank Olken and Doron Rotem
Computer Science Research Dept.

Information and Computing Sciences Div.
Lawrence Berkeley Laboratory
1 Cyclotron Road, Berkeley, CA 94720

## Abstract

We consider the design and analysis of algorithms to retrieve simple random samples from databases. Specifically, we examine simple random sampling from $B^+$ tree files. Existing methods of sampling from $B^+$ trees, require the use of auxiliary rank information in the nodes of the tree. Such modified $B^+$ tree files are called "ranked $B^+$ trees". We compare sampling from ranked $B^+$ tree files, with new acceptance/rejection (A/R) sampling methods which sample directly from standard $B^+$ trees. Our new A/R sampling algorithm can easily be retrofit to existing DBMSs, and does not require the overhead of maintaining rank information. We consider both iterative and batch sampling methods.

## 1 Introduction

Virtually all database systems used to record financial transactions (accounting systems, inventory control systems, bank records, etc.) are subject to annual audit, usually involving random sampling of the records for corroboration. Yet commercial database management systems do not support queries to retrieve a random sample of some portion of the database. One reason is that previous proposals to support retrieval of random samples from databases have required the modification of standard access methods, and the maintenance of additional information in the indices. In particular, earlier authors have described sampling from modified $B^+$ trees, *called ranked $B^+$ trees*, which incorporate information which permits the computation of the rank of a record.

In this paper we discuss methods of sampling from regular $B^+$ trees, employing acceptance/rejection (A/R) methods. These new algorithms do not require any modification of the standard $B^+$ tree structures, nor do they require maintenance of any additional fields in the $B^+$ trees. Hence these new sampling methods can be more easily retrofit to existing DBMSs. While not quite as efficient as earlier proposals, it should be suitable for applications which only need sampling infrequently, e.g. for auditing. For $B^+$ tree files we discuss both iterative and batch sampling methods.

### 1.1 Why sample?

Random sampling is used on those occasions when processing the entire dataset is not necessary and is considered too expensive in terms of response time or resource usage. The savings generated by sampling may arise either from reductions in the cost in retrieving the data from the DBMS or from subsequent "post-processing" of the sample.

Retrieval costs are significant when dealing with large administrative or scientific databases. Post-processing of the sample may involve expensive statistical computations, or further physical examination of the real world entities described by the sample. Examples of the latter include physical inspection and/or testing of components for quality control [Mon85,LWW84], physical audits of financial records [Ark84,LTA79], and medical examinations of sampled patients for epidemiological studies. As noted above, most financial databases are subject to annual audits, which typically entail random sampling of records from the database for corroboration.

Sampling is useful for applications which are attempt-

*This is a condensed version of tech report LBL–25517. This work was supported by the Director, Office of Energy Research, Office of Basic Energy Sciences, Applied Mathematical Sciences Division of the U.S. Department of Energy under Contract DE-AC03-76SF00098. Authors electronic mail addresses: olken@csam.lbl.gov, olken@lbl.bitnet, rotem@csam.lbl.gov



**Proceedings of the Fifteenth International Conference on Very Large Data Bases**

**Amsterdam, 1989**

ing to estimate some aggregate property of a set of records, such as the total number of records which satisfy some predicate. Thus random sampling is typically used to support statistical analysis of a dataset, either to estimate parameters of interest [HOT88] or for hypothesis testing. See [Coc77] for a classic treatment of the statistical methodology. Applications include scientific investigations such as high energy particle physics experiments, quality control, and policy analyses. For example, one might sample a join of welfare recipient records with tax returns or social security records in order to estimate welfare fraud rates.

### 1.2 Simple Random Sampling

There are a variety of types of sampling which may be performed. In this paper we shall be concerned with *fixed size* samples, where the sample size has been specified by the user.

Another way of characterizing the sample is whether it is drawn with or without replacement. Let us assume that we are sampling from a distinct set of records. Samples drawn with replacement may contain duplicates, and are usually easier to obtain (as we shall see). However, samples drawn without replacement, which proscribe duplicates, generally provide more information for a given sample size.

If the inclusion probabilities for individual records are uniform we say that we have a *simple random sample (SRS)*.

In this paper we will deal with fixed size simple random samples. One can readily convert between simple random samples with and without replacement (denoted SRSWR and SRSWOR respectively). To convert from SRSWR to SRSWOR we merely remove duplicates (in time linear in the sample size via a hash table) (possibly we must increase the sample size to compensate for duplicate removals). To convert from SRSWOR to SRS we must generate synthetic duplicates, this also can be done in time linear with the sample size. If the sampling fraction (ratio of sample size to population size) is (as is typical) small, then there will be few duplicates in the SRSWR and the extra samples needed to compensate for duplicate removal will be insignificant. We shall ignore the costs incurred due to duplicate removal and replacement in this paper.

### 1.3 Notation and Efficiency Metric

In this paper we shall measure the efficiency of the sampling algorithm in terms of the number of disk blocks read. We have typically assumed that the desired sample size is much smaller than the number of records in the file, (hence the likelihood of duplicate records in the sample is small) so that we have ignored the difference between sampling with and without replacement. A table of frequently used notation follows.

## 2 Basic Techniques

### 2.1 Acceptance/Rejection Sampling

A basic tactic used in this paper is acceptance/rejection sampling. It can be used to construct *weighted samples* in which the inclusion probabilities of a record are proportional to some arbitrary weight (calculated from the record attributes). In this paper we will use it to compensate for algorithm or data structure induced variations in sample inclusion probabilities so as to finally obtain a *simple random sample*, i.e., one with uniform inclusion probabilities. A brief explanation of this classic sampling technique is included here for those in the database community who may be unfamiliar with it.

Suppose that we wish to draw a weighted random sample of size 1 from a file of $N$ records, denoted $r_j$, with inclusion probability for record $r_j$ proportional to the weight $w_j$. The maximum of the $w_j$ is denoted $w_{max}$.

We can do this by generating a uniformly distributed random integer, $j$, between 1 and $N$, and then accepting the sampled record $r_j$ with probability $p_j$:

$$p_j = \frac{w_j}{w_{max}} \tag{1}$$

The acceptance test is performed by generating another uniform random variate, $u_j$, between 0 and 1 and accepting $r_j$ if $u_j < p_j$. If $r_j$ is rejected, we repeat the process until some $j$ is accepted.

The reason for dividing $w_j$ by $w_{max}$ is to assure that we have a proper probability (i.e., $p_j \leq 1$). If we do not know $w_{max}$ we can use instead a bound $\Omega$ such that $\forall j, \Omega > w_j$,. The number of iterations required to accept a record $r_j$ is geometrically distributed with a mean of $(E[p_j])^{-1}$. Hence using $\Omega$ in lieu of $w_{max}$ results in a less efficient algorithm.

Acceptance/rejection sampling is well suited to sampling with *ad hoc* weights or when the weights are being frequently updated (since it does not require auxiliary indices).

### 2.2 A Review of Sampling from Files

Over the last 20 years there has been considerable work done on developing basic techniques for sampling from a single flat file (usually with fixed blocking). We employ some of these techniques in our work on query sampling. In Table 3 we list the major results, with citations to the relevant algorithms.

| | |
|---|---|
| $\alpha_k$ | acceptance probability of record $k$ |
| $\alpha$ | $= E(\alpha_k)$ = expected acceptance probability of record |
| $\beta$ | $= (f_{avg}/f_{max})$ = average acceptance prob. at of a node |
| $b_{avg}$ | average blocking factor for variably blocked file |
| $b_{max}$ | maximum blocking factor for variably blocked file |
| $C_{method}(s)$ | cost of retrieving sample of size $s$, via specified method |
| $f_i$ | fan-out from internal node $i$ of $B^+$ tree, or<br>number of records in leaf node $i$ of $B^+$ tree |
| $f_{avg}$ | average fan-out from internal node of $B^+$ tree, or<br>average number of records in leaf node of $B^+$ tree |
| $f_{max}$ | maximum fan-out from internal node of $B^+$ tree, or<br>maximum number of records in leaf node of $B^+$ tree |
| $h$ | height of $B^+$ tree(count root as height 1) |
| $h'$ | height of ranked $B^+$ tree<br>(usually same as $h$ ) |
| $i = 0$ | denotes root node of $B^+$ tree |
| $n$ | number of records in file |
| $p_k$ | probability of inclusion of record $k$ |
| $path_k$ | path (node identifiers) from root to leaf containing record $k$ |
| $\pi$ | expected length of path in early abort algorithm |
| $s$ | number of records desired in sample |
| $s'$ | inflated sample size (to compensate for acceptance/rejection) |
| $thisnode$ | pointer to a $B^+$ tree node (internal or leaf) |
| $Y(k, m)$ | Cardenas's function for expected number of blocks<br>referenced when retrieving $k$ records from $m$ block file |
| $w_k$ | probability of sampling of record $k$ on a simple random walk from root to leaf of $B^+$ tree |

Table 1: Notation used in the paper.

| | |
|---|---|
| ARHASH | A/R algorithm for hashed files |
| NI | Naive Iterative algorithm for $B^+$ tree files |
| EAI | Early Abort Iterative algorithm for $B^+$ tree files |
| RI | Iterative algorithm for ranked $B^+$ tree files |
| NB | Naive Batch algorithm for $B^+$ tree files |
| EAB | Early Abort Batch algorithm for $B^+$ tree files |
| RB | Batch algorithm for ranked $B^+$ tree files |

Table 2: Algorithm abbreviations

| Type of sampling | Citation | Expected Disk Accesses |
|---|---|---|
| SRSWR | [Yao77] | $O(s)$ |
| SRSWR, variable blocking | this paper | $O(s(b_{max}/b_{avg}))$ |
| SRSWOR | [EN82] | $O(s)$ |
| Weighted RS | [WE80] | $O(s \log n)$ |
| Sequential RS, known pop. size | [FMR62] | $O(n/b_{avg})$ |
| | [Vit84] | $O(s)$ |
| Sequential RS, unknown pop. size | | $O(n/b_{avg})$ |
| | [Vit85] | $O(s(1 + \log(n/s))$ |

Table 3: Basic Sampling Techniques from a single file. Assume each sample taken from a distinct disk page, i.e., $s \ll (n/b_{avg})$ For Vitter's algorithms assume random disk I/O.

# 3 Iterative Sampling from a $B^+$tree

## 3.1 A/R Sampling from a $B^+$ tree

In this section we discuss how to employ acceptance/rejection sampling to sample from a $B^+$ tree without requiring the storage of any additional information in the $B^+$ tree nodes. Although our new method has a higher retrieval cost than earlier methods based on ranked $B^+$ trees, it does not require any modification of existing access methods, nor any additional update costs. Hence this method will be preferred over earlier methods for applications where updates dominate sampling retrievals.

For expository reasons we commence with a discussion of the *naive method* (a random walk from root to leaf, followed by an acceptance/rejection test). Subsequently, we show that a modification of this method, known as *early abort*, dominates the *naive method*. In Section 4 we consider *batch* versions of each algorithm, which in turn dominate the original *iterative* algorithms discussed in this section.

We shall assume that the buffer pool is sufficiently large to cache one entire path (from root to leaf) of the $B^+$ tree. For simplicity of analysis, we neglect the minor effect of caching beyond the root page (for these iterative algorithms). This will not alter the relative performance of the various iterative algorithms.

### 3.1.1 The Problem

As with other file structures the problem is to produce uniform inclusion probabilities for the target data records. Simply choosing a random edge from each internal node will not suffice, because nodes reached from internal nodes with low fanout will be more likely to be sampled than those reached from nodes with high fanout.

### 3.1.2 Naive method

Basically, the naive method consists of performing acceptance/rejection sampling on complete random paths through the tree (from root to leaf). The acceptance/rejection sampling is used to correct the inclusion probability of each sampled path, so that every record (stored in the leaves of the $B^+$ tree) has the same inclusion probability. We dicuss this method primarily for expository reasons, since (as we shall show) it is dominated by the *early abort method* (described in Sect. 3.1.3).

In this method we select a random path from the root to a record in a leaf (i.e., at each internal node we choose a branch at random (equi-probably), at the leaf we select a record at random (equi-probably)). Upon reaching the leaf we perform an acceptance/rejection test to decide whether to keep this path. The acceptance probability is calculated as we traverse the path from root to leaf as the product of the ratios of actual fan-out to maximum fan-out at each node (except the root). We are in effect sampling from a full multi-way tree, discarding paths which do not actually exist.

We denote by $B$ a $B^+$ tree of order $m$ with height $h$ (there exist $h$ nodes on any path from root to leaf, including the root and leaf). Let $f_i$ denote the fan-out of node $i$. This is the number of branches from an internal node, and the number of records in a leaf node. We designate the root node to be node zero.

**Lemma 1** *The naive algorithm generates a simple random sample. The inclusion probability $p_k$ for record $r_k$ contained in leaf $j$ for a single (root-to-leaf) path traversal is:*

$$p_k = {f_0}^{-1} {f_{max}}^{-h+1} \tag{2}$$

*where $f_{max} = 2m+1$ is the maximum fanout of an internal node (and for simplicity also the maximum number of records on a leaf node).*

**Proof:** Let $w_k = p$(sampling record $k$ on a random walk). Then

$$w_k = \prod_{i \in path_k} f_i^{-1} \tag{3}$$

where $path_k$ refers to the path from root to leaf node containing the record $k$. Let the acceptance probability for record $k$ be $\alpha_k$, defined:

$$\alpha_k = \prod_{i \in path_k, i \neq 0} (f_i / f_{max}) \tag{4}$$

Note that the product here excludes the root node, because it is common to all paths, hence it does not introduce any non-uniformity into the inclusion probabilities. Recall that all paths in the tree have the same height, $h$. Hence:

$$p_k = \alpha_k w_k = f_0^{-1} \prod_{i \in path_k, i \neq 0} (1/f_{max}) \tag{5}$$

$$p_k = {f_0}^{-1} {f_{max}}^{-h+1} \tag{6}$$

□

**Theorem 1** *The expected cost of the naive method for a simple random sample with replacement of size $s$ from a $B^+$ tree is approximately:*

$$E[C_{NI}(s)] \approx (\beta^{-1})^{h-1} s(h-1) + 1 \tag{7}$$

**Proof:** Caching is generally only effective for the root page, hence the $(h-1)$ factor, rather than $h$. The last term simply accounts for initially reading the root page. Hence:

$$E[C_{NI}(s)] \approx s'(h-1)+1 \tag{8}$$

where $s'$ is the gross sample size necessary to produce a net sample of size $s$ after acceptance/rejection sampling. By assuming that all the fan-outs are equal to $f_{avg}$ we have:

$$E[s] \approx s'\beta^{h-1} \tag{9}$$

Inverting this equation we shall assume (ignoring stochastic variation) that the required gross sample size, $s'$ is:

$$s' = (\beta^{-1})^{h-1}s \tag{10}$$

Substituting yields the theorem. □

### 3.1.3 Early abort method

The *early abort method* of sampling from a $B^+$ tree, derives from the *naive* method. Both are based on acceptance/rejection sampling of random paths in the tree. The difference is that the naive method traverses complete paths from root to leaf before deciding on acceptance or rejection while the *early abort method* performs an acceptance/rejection test at each node (except the root).

If the node is rejected, then we can abort searching this path, permitting early abortions of a path. We can in effect reject a leaf node while part way down the path to it, without requiring that we retrieve the entire path. Hence, this method clearly dominates the naive method in expectation.

One way of thinking about the algorithm is to imagine that we are sampling random paths from the full multiway tree. As soon as we go down a branch which is not present in the actual (partially full) tree, we abort that path.

At each node (except the root) along a path from root to leaf we perform an acceptance/rejection test with acceptance probability for node $i$ denoted as $\beta_i$:

$$\beta_i = f_i/f_{max} \tag{11}$$

Recall

$$\beta = f_{avg}/f_{max} \tag{12}$$

The root node is accepted unconditionally, i.e., $\beta_0 = 1$.

**Lemma 2** *For a single (root-to-leaf) path traversal, the early abort algorithm generates a simple random sample with inclusion probability $p_k$ for record $r_k$, where:*

$$p_k = {f_0}^{-1}{f_{max}}^{-h+1} \tag{13}$$

**Proof:** Observe that the probability, $b_k$, of accepting a record $k$ (in a leaf node) is simply the product of the $\beta_i$ along the path:

$$b_k = \prod_{i \in path_k, i \neq 0} \beta_i = \alpha_k \tag{14}$$

i.e., the same as for the *naive* algorithm. Hence our result follows from the proof of the naive algorithm. □

In effect, the early abort algorithm searches the same paths as the naive algorithm, but it aborts the search at the first rejection. Thus for some paths, it accesses fewer pages. Hence its expected cost will be strictly less than that of the naive algorithm, unless all of the pages are full.

**Theorem 2** *The expected cost of the early abort iterative method for a simple random sample with replacement of size s from a $B^+$ tree is approximately:*

$$E[C_{EAI}(s)] \approx (\beta^{-1})^{h-1}s\frac{(\beta^{h-1}-1)}{\beta-1}+1 \tag{15}$$

**Proof:** Here we have again assumed that all the fan-outs are equal to $f_{avg}$. As before, the gross sample size must be increased by a factor of $(\beta^{-1})^{h-1}$ to account for the losses due to A/R sampling.

Recall that for the naive algorithm the length of each path examined is $h$ (with a one-path cache it will be approx. $h-1$). What is the expected length of a path searched in the early abort algorithm? Let $\pi$ denote the average path length (assuming the root is cached). Then we ignore the root (because it is cached). Acceptance/rejection sampling at each node could cause an early abort - but this happens after the node has been read in - so we subtract one more from the exponent on the expected acceptance probability $\beta = (f_{avg}/f_{max})$ of a node. This gives:

$$\pi = \sum_{i=1}^{i=h-1} \beta^{i-1} \tag{16}$$

Summing we have:

$$\pi = \frac{(\beta^{h-1}-1)}{\beta-1} \tag{17}$$

The total cost is given by:

$$E[C_{EAI}(s)] \approx s'\pi + 1 \tag{18}$$

where the last term consists of reading the root and where again we have

$$s' = s(\beta^{-1})^{h-1} \tag{19}$$

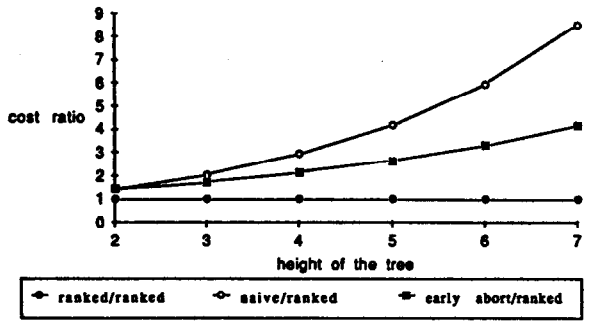


Figure 1: Iterative Algorithm Performance Graph

is the adjusted gross sample size to compensate for the attrition due to acceptance/rejection sampling. Substituting yields the theorem. □

Observe that from our definition of $\pi$ that it must be less than $h-1$, hence we have:

**Theorem 3** *The expected cost of the early abort iterative method is strictly less that the expected cost of the naive iterative method.*

**Proof:** Both methods must explore the same expected number of paths, as noted above the expected path length for early abort is less than that for the naive algorithm. □

We summarize the performance of the iterative algorithms in Figure 1.

### 3.2 Sampling from a ranked $B^+$ tree

In this section we discuss how to extract a fixed size simple random sample from a *ranked* $B^+$ tree file. A *ranked* $B^+$ *tree* file is one whose nodes have have been augmented with information which permits one to find the $j'th$ record in the file. We include this method as a benchmark against which to compare our new algorithms.

Suppose that we wish to sample from a file containing $n$ records. We generate a uniformly distributed random number, $j$, between 1 and $n$, and then sample the $j'th$ record. To do this we must be able to identify the $j'th$ record. If our access method to the file is a $B^+$ tree, then we must be able to find the $j'th$ ranked record in the file. Hence we must store information in the tree which allows one to calculate the rank of each record.

This idea is discussed in [Knu73]. Similar ideas are used in [BK75] and [WE80]. Essentially we store in each node of the tree a count (partial sum) of the number of leaves in that subtree. In a binary tree [Knu73] the rank of each leaf can be calculated by suitable sums and differences of the count fields of all the nodes on the path from root to leaf. In a $B^+$ tree we promote the count fields one level in the tree so that each node stores not only the total count of leaves in its subtree, but also the counts for each child (alongside its key). Hence, while a rank access to the tree must still examine on average half the entries in each $B^+$ tree page, the number of disk pages which must be fetched is only equal to the height of the tree.

This matter is discussed in [SL88], and (for tries) in [Gho86].

For this algorithm, which we call the ranked iterative algorithm (denoted RI), we simply generate a random number, $j$, between 1 and $n$ (the total number of records in the file) and then access the $B^+$ tree via the rank fields to retrieve the $j'th$ record. If there is no caching of disk pages, then each random probe retrieves a complete path from root to leaf, consisting of $h'$ pages where $h'$ is the height of the ranked $B^+$ tree.

**Theorem 4** *The expected number of disk pages accessed for a simple random sample with replacement of size $s$ from a ranked $B^+$ tree via the ranked iterative algorithm RI is:*

$$E(C_{RI}(s)) \approx s(h'-1)+1 \qquad (20)$$

*where $h'$ is the height of the ranked $B^+$ tree, i.e., $h' = h/(1+log_{f_{avg}}(2/3))$, assuming that storing the rank field reduces the fan-out of internal nodes by 1/3.*

**Proof:** Proof omitted. See full paper. □

In practice, the ranked $B^+$ tree will have a height no more than one greater, than the corresponding $B^+$ tree.

As discussed earlier, duplicate removal can be performed in $O(s)$ memory. By checking online for duplicate random numbers, before fetching each record, we can obtain a SRS without replacement in the same number of disk accesses.

## 4 Batch Sampling from $B^+-trees$

In this section we consider batch methods of sampling from $B^+$ trees. Such methods are intended to reduce or eliminate the re-reading of disk blocks incurred by iterative sampling algorithms assuming that the buffer pool can hold one entire path through the $B^+$ tree. They process the entire sample as one batch, proceeding from one end of the file to the other. Batch sampling from ranked $B^+$ trees completely eliminates re-reading

disk blocks. Batch sampling of regular $B^+$ trees via acceptance/rejection methods returns a sample of random (i.e., binomial) size. Hence, it may occassionally be necessary to repeat the process to obtain a sufficiently large sample.

## 4.1 Standard $B^+$ trees

### 4.1.1 Naive Batch

The naive batch sampling algorithm for standard $B^+$ trees simply applies the naive iterative algorithm in parallel. Instead of sampling paths through the tree one at a time, we process a batch of paths at once. The advantage is that we reduce the number of times pages must be rereferenced.

Suppose that we have an estimate $s'$ of the required gross sample size needed to produce a net sample of size $s$. We start at the root with a gross sample size of $s'$ and proceed with a depth first search of the tree.

At each internal node of the tree we allocate the incoming portion of the sample to the various subtrees by generating a multinomial random vector. A multinomial random vector from a population of size $s$ in $k$ cells is a vector $V = (v_1, v_2, ..., v_k)$ of length $k$, which records the number of balls $v_i$ which fall in cell $i$, when $s$ balls are thrown in the the cells at random (equiprobably). (More generally the probability of balls falling into each cell could vary, but we do not need this.) The uniform multinomial vector can be generated in two ways: by generating a random branch for each of $s$ balls and incrementing the corresponding cell count, or alternatively by generating a Poisson random variable for each cell, and then adjusting the resulting variable with the first method, so that their sum is correct.

Only those branches with nonzero sample sizes allocated to them are pursued.

Upon reaching a leaf node we perform acceptance/rejection sampling on the portion of the gross sample allocated to this leaf. This we do by generating a binomial random variable, $x_k \sim B(s_k, \alpha_k)$ with parameters, $s_k$ = gross sample size allocated to this leaf, and $\alpha_k$ =acceptance probability for this path $k$ (as in naive iterative algorithm).

Having determined the net sample size for a particular leaf, we extract a simple random sample with replacement of this size from the records on the leaf (this is trivial).

We then continue with the depth first search of the tree until complete. The resulting sample may be the wrong size (because of acceptance/rejection). If the resulting sample is too large, we discard the excess (chosen at random). If the resulting sample is too small, we repeat the process (adding to our sample) until we have enough.

**Lemma 3 Cardenas's Lemma** *The expected number of disk blocks referenced, d, when sampling k records (with replacement) from a file of m equal size blocks is given by:*

$$d = m(1 - (1 - (1/m)^k)) \tag{21}$$

*We will denote this function of m and k as $Y(k, m)$.*

**Proof:** See [Yao77]. □

For the naive batch method, the effort to retrieve a gross sample of size $s'$, is the same as the effort to perform batch searching on a $B^+$ tree, with the same batch size.

**Theorem 5** *The expected cost in I/O to retrieve a sample of size s via the naive batch sampling method is approximately:*

$$E(C_{NB}(s)) \approx 1 + \sum_{j=1}^{j=h-1} Y(s', f_0 f^{j-1}) \tag{22}$$

*where $f = f_{avg}$ and and $s' = s(\beta^{-1})^{h-1}$ (inflated gross sample size) as before.*

**Proof:** As before, we use a gross sample size of $s'$ to compensate for the losses due to acceptance rejection. This theorem is a derived by applying Cardenas's Lemma (Lemma 3) for each level of the tree. We have assumed that all nodes have fan-out $f = f_{avg}$, except the root which has fan-out $f_0$.

Let us number the levels 0, 1, 2, ..., h-1 from root to leaf, and let $j$ denote the level number.

For level 0, we have $s'$ records, 1 block (the root). For level 1, we still have $s'$ records, and $f_0$ blocks. For level 2, we have $s'$ records, and $f_0 f$ blocks. Thus for level $j$, $j > 1$, we have $s'$ records and $f_0 f^{j-1}$ blocks, hence by Cardenas's Lemma the number of blocks accessed at level $j$ will be $Y(s', f_0 f^{j-1})$. Summing yields the theorem. □

### 4.1.2 Early Abort Batch Method

The early abort batch method is simply the batch analog of the early abort iterative method. It is identical to the naive batch method, except that the acceptance/rejection sampling is performed (by computing binomial samples) at each node (except the root) as we search from root to leaf. As was the case with the iterative methods, the early abort batch method dominates the naive batch method.

Thus we commence our depth first search at the root with a gross sample size of $s'$. At the root, and each subsequent internal node, we allocate the incoming sample to the various branches by means of a multinomial random vector. Only those branches with nonzero sample sizes allocated to them are pursued.

At each level beyond the root, we perform acceptance/rejection sampling of the incoming sample by generating a binomial random variable, $x_i \sim B(s_i, \beta)$ with parameters $s_i$, (the incoming sample size), and acceptance probability $\beta_i$ (as in the iterative early abort algorithm). The resulting net sample size for the node is then allocated to the branches via a multinomial random vector. Only those branches with nonzero sample sizes allocated to them are pursued.

Upon reaching a leaf we perform one last acceptance/rejection test (by generating a binomial random variable, $x_i \sim B(s_i, \beta)$) to account for the variable loading of leaf pages. We then extract a simple random sample with replacement of size $x_i$ of the records on the page.

Incorrect net sample sizes are dealt as described above for the naive batch sampling algorithm.

For the early abort batch method, the analysis is more complicated. The search paths which are aborted early do not generate as many page accesses. If we number the levels from 0 to $h$, denoting by *level* $i$ the node at distance $i$ from the root, then at level $i$, $i > 0$, we will access $k_i$ records from $f_0 f_{avg}^{\,i-1}$ pages, where $k_i$ is approximately a binomial random variable with expectation $s' \cdot \beta^{i-1}$, where we have ignored the variation in $f_i$, replacing it with $f_{avg}$.

**Theorem 6** *The expected I/O cost to retrieve a simple random sample of size $s'$ via the early abort batch sampling method is approximately:*

$$E(C_{EAB}(s)) \approx 1 + \sum_{j=1}^{j=h-1} Y(s'\beta^{j-1}, f_0 f^{j-1}) \qquad (23)$$

*where $f = f_{avg}$ and $s' = s(\beta^{-1})^{h-1}$ (inflated gross sample size) as before, and Y(k,m) is Cardenas's function defined above.*

**Proof:** As before, we use a gross sample size of $s'$ to compensate for the losses due to acceptance/rejection. This theorem is derived by applying Cardenas's Lemma (Lemma 3) for each level of the tree. We have assumed that all nodes have fan-out $f = f_{avg}$, except the root which has fan-out $f_0$.

Let us again number the levels 0, 1, 2, ..., h-1 from root to leaf, and let $j$ denote the level number.

For level 0, we have $s'$ records, 1 block (the root). For level 1, we still have $s'$ records, and $f_0$ blocks. For level 2, we have $s'\beta$ records (because we have done acceptance/rejection at level 1), and $f_0 f$ blocks. Thus for level $j$, $j > 1$, we have $s'\beta^{j-1}$ records and $f_0 f^{j-1}$ blocks, hence by Cardenas's Lemma the number of blocks accessed at level $j$ will be $Y(s'\beta^{j-1}, f_0 f^{j-1})$. Summing yields the theorem. □

## 4.2 Ranked $B^+$ trees

Finally, we consider batch sampling of ranked $B^+$ trees. We simply generate a simple random sample of the ranks, sort it, and then perform a batch search of the ranked $B^+$ treefile. Alternatively we could use Vitter's algorithm [Vit84] to generate the sequential skips required to determine the ranks of the sampled records. (Note that Vitter's algorithm generates a simple random sample without replacement, rather than with replacement. This is easily corrected.) In any case we are only concerned here with I/O, and we assume the sample of ranks can easily fit in memory.

We shall assume that we have a cache large enough to hold a complete path through the tree from root to leaf, so that reexamining pages along this path required to retrieve the sample is costless (in terms of disk I/O). Then retrieval of the sample is equivalent to batch searching of a $B^+$ tree, a classic problem treated by [Pal85] (among others). Essentially, the number of pages to be retrieved is simply the number of distinct pages in the union of all paths to sampled pages.

**Theorem 7** *The expected I/O cost to retrieve a simple random sample of size s from a $B^+$ tree via the ranked batch sampling method is approximately:*

$$E(C_{NB}(s)) \approx 1 + \sum_{j=1}^{j=h-1} Y(s, f_0 f^{j-1}) \qquad (24)$$

*where $f = f_{avg}$, $f_0$ is the fan-out of the root, and $Y(k,m)$ is Cardenas's function defined above.*

**Proof:** The proof is identical to that of Theorem 5, except that we do not need to inflate the sample size since we are not doing acceptance/rejection sampling. □

## 4.3 Comparisons

It is clear the batch algorithms will dominate the respective iterative algorithms, because they avoid rereading disk blocks which are used in more than one sample path.

Furthermore:

**Theorem 8** *The early abort batch outperforms naive batch.*

$$E[C_{EAB}(s)] \leq E[C_{NB}(s)] \qquad (25)$$

**Proof:** This result follows from the cost functions and the fact that Carndenas's functions $Y(k,m)$ is monotone increasing in $k$ (the sample size). □

The relationship between the performance of the ranked batch algorithm and the early abort batch algorithm is more subtle. Because the ranked batch algorithm may increase the height of the $B^+$ tree it is possible

that RB will perform worse than EAB. However, more typically RB will not increase the height of the $B^+$ tree and will out-perform EAB. How big is the difference between $E[C_{RB}(s)]$ and $E[C_{EAB}(s)]$? It is clearly less than a factor of $(\beta^{-1})^{h-1}$, the factor by which we inflate the gross sample size of EAB to compensate for the attrition due to acceptance/rejection sampling. For random $B^+$ trees $\beta$ is known to be 0.7. Hence for a random $B^+$ tree of height 5, the methods differ by a factor of no more than 4.

# 5 Conclusions

Our most important results concern algorithms to retrieve simple random samples of $B^+$ trees, without any additional data structures or indices. These methods are based on acceptance/rejection sampling, and provide a simple, inexpensive way to add sampling to a relational database systems. They are appropriate for systems which only infrequently need to support sampling, e.g., for auditing. The batch early abort algorithm offers the best performance of algorithms of this class, and is to be generally preferred to the iterative algorithms.

## Acknowledgements

The authors would like to thank Tekin and Meral Ozsoyoglu for their continuing encouragement. Jack Morgenstein first introduced us to the problem. We also wish to thank the referees for their comments.

# References

[Ark84] Herbert Arkin. *Handbook of Sampling for Auditing and Accounting.* McGraw-Hill, 1984.

[BK75] B.T. Bennett and V.J. Kruskal. Lru stack processing. *IBM Journal of Research and Development*, 19(4):353–357, July 1975.

[Coc77] William G. Cochran. *Sampling Techniques.* Wiley, 1977.

[EN82] Jarmo Ernvall and Olli Nevalainen. An algorithm for unbiased random sampling. *The Computer Journal*, 25(1), 1982.

[FMR62] C.T. Fan, M.E. Muller, and I. Rezucha. Develoment of sampling plans by using sequential (item by item) selection techniques and digital computers. *Journal of the American Statistical Association*, 57:387–402, June 1962.

[Gho86] S. Ghosh. Siam: statistics information access method. In *Proceedings of the Third International Workshop on Statistical and Scientific Database Management*, pages 286–293, EUROSTAT, Luxembourg, 1986.

[HOT88] Wen-Chi Hou, Gultekin Ozsoyoglu, and Baldeo K. Taneja. Statistical estimators for relational algebra expressions. In *Proceedings of the Seventh ACM Conference on Principles of Database Systems*, pages 288–293, March 1988.

[Knu73] Donald Ervin Knuth. *The Art of Computer Programming: Vol. 3, Sorting and Searching.* Addison-Wesley, 1973.

[LTA79] Donald A. Leslie, Albert D. Teitlebaum, and Rodney J. Anderson. *Dollar Unit Sampling.* Copp Clark Pitmanan, 1979.

[LWW84] H.-J. Lenz, G.B. Wetherill, and P.-Th. Wilrich, editors. *Frontiers in Statistical Quality Control 2.* Physica-Verlag, Wurzburg, Germany, 1984.

[Mon85] Douglas C. Montogmery. *Introduction to Statistical Quality Control.* Wiley, 1985.

[Pal85] P. Palvia. Expressions for batched searching of sequential and hierarchical files. *ACM Transactions on Database Systems*, 10(1):97–106, March 1985.

[SL88] J. Srivastava and V.L. Lum. A tree based access method (tbsam) for fast processing of aggregate queries. In *Proceedings of the 4th International Conference on Data Engineering*, pages 504–510, IEEE Computer Scoeity, 1988.

[Vit84] Jeffrey Scott Vitter. Faster methods of random sampling. *Communications of the ACM*, 27(7):703–718, July 1984.

[Vit85] Jeffrey Scott Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software*, 11(1):37–57, March 1985.

[WE80] C.K. Wong and M.C. Easton. An efficient method for weighted sampling without replacement. *SIAM Journal on Computing*, 9(1):111–113, February 1980.

[Yao77] S. Bing Yao. Approximating the number of accesses in database organizations. *Communications of the ACM*, 20(4):260–261, April 1977.