Random Sampling from Databases

Adrian Moos
adrian.moos@student.ethz.ch

18th July 2005
Contents

0.1 Preface ............................................. 3

1 Introduction ........................................... 3
  1.1 Why Sample? ........................................ 3
    1.1.1 Statistical Data Analysis .......................... 3
    1.1.2 Obtaining Representative Samples .................. 4
    1.1.3 Our Goal: Obtaining Random Samples .............. 5
    1.1.4 Applications ..................................... 5
  1.2 Definitions ......................................... 5
    1.2.1 Random Sample ................................... 5
    1.2.2 Classification of Algorithms ...................... 5

2 Sampling from Indices .................................. 6
  2.1 Basic Tools .......................................... 6
    2.1.1 Introductory Exercise .............................. 6
    2.1.2 A/R Sampling ..................................... 7
    2.1.3 Effect of A/R Sampling on Sample Distribution ..... 7
    2.1.4 Efficiency of A/R Sampling ......................... 8
  2.2 Sampling from $B^+$ trees / Early Abort ............. 8
    2.2.1 Genesis .......................................... 8
    2.2.2 The Algorithm ................................... 9
    2.2.3 Correctness ..................................... 9
    2.2.4 Efficiency ..................................... 9
  2.3 Sampling from Ranked Trees .......................... 11
    2.3.1 Method .......................................... 11
    2.3.2 Comparison to Early Abort ......................... 11

3 Sampling from Queries .................................. 12
  3.1 Selection ........................................... 12
    3.1.1 Problem Statement ................................. 12
    3.1.2 First Thought ..................................... 12
    3.1.3 Without Index ................................... 12
    3.1.4 With Index ...................................... 13
    3.1.5 Summary ......................................... 13
  3.2 Further Work ........................................ 13

4 Conclusion ............................................. 14
0.1 Preface

This text accompanies my seminar presentation on the same topic. It is primarily based on Frank Olken’s Random Sampling from Databases Ph.D. dissertation, UC Berkeley, April 1993, LBL Technical Report 32883 which at the time of writing can be found at http://pueblo.lbl.gov/~olken/sampling.html

Please let me know if you find any mistakes.

1 Introduction

1.1 Why Sample?

1.1.1 Statistical Data Analysis

In statistical data analysis, we are often interested in some aggregate property over a set (=population) of records, e.g.

- what is the average income of a swiss citizen?
- what proportion of ETH Students reads 20 Minuten daily?

Although such questions are often highly relevant in describing the workings or the state of larger systems, the straightforward way to answer them (i.e. obtaining data about every member of the population, then compute the aggregate) is often too expensive. Instead, a cheaper (in price, not capability) method is used:

1. choose a random subset of the population (called sample)
2. aggregate over the sample instead of the population

Clearly, this method requires that

\[ \text{aggregate(sample)} \approx \text{aggregate(population)} \]

We call a sample representative iff this is the case.

Now that we know we want representative samples, we turn our attention to how they can be obtained. At first glance this might seem difficult, but the following proof reveals that for many aggregates, there is a reasonably small sample size such that the vast majority of samples of that size is representative – consequently, we can simply use a random sample of that size, knowing that it will be representative with high probability.
1.1.2 Obtaining Representative Samples

Often, the aggregate can be expressed as the average of a property on elements:

$$\text{aggregate}(A) \overset{def}{=} \frac{1}{|A|} \sum_{a \in A} f(a) = E[f(A^*)]$$

where $A^*$ is a random variable denoting a uniformly random element from $A$. Let $A_i$ be the elements in the random sample.

We will now prove that in this setting, there is a sample size $s$ such that a random sample of size $s$ is representative with high probability.

Our definition of a random sample \(^1\) will imply that the $A_i$ are independent and uniformly distributed.

For convenience of notation, let $X \overset{def}{=} f(A^*)$ and $X_i \overset{def}{=} f(A_i)$. Note that the $X_i$ are independent and distributed like $X$. Because $A$ is finite, so are $E[X]$ and $Var[X]$.

**Theorem 1 (Weak Law of Large Numbers)** if $X_i$ is a family of independently identically distributed random variables, $\mu = E[X_i]$ and $\sigma^2 = Var[X_i]$ are finite and $\varepsilon > 0$, then:

$$\lim_{s \to \infty} P \left[ \left| \frac{1}{s} \sum_{i=1}^{s} X_i - E[X_1] \right| \leq \varepsilon \right] = 1$$

As noted above, the $X_i$ satisfy the requirements of the theorem.

Also, we have:

$$\text{avg}(\text{sample}) = \frac{1}{s} \sum_{i=1}^{s} X_i$$

$$\text{avg}(\text{population}) = E[X]$$

We conclude:

$$\lim_{s \to \infty} P [|\text{avg}(\text{sample}) - \text{avg}(\text{population})| \leq \varepsilon] = 1$$

which implies that for every probability of success $\delta < 1$ and every margin of error $\varepsilon > 0$ there is a sample size $s$ such that the average of the sample is within $\varepsilon$ of the population’s average with probability at least $\delta$.

We have proven that if the aggregate is an average of some element-property over a finite set, that, for every probability of success $< 1$, there is a sample size that is representative with that probability. Statistical literature contains similar arguments for other kinds of aggregates (e.g. percentiles) and also upper bounds for the required sample size, the gist of which being that the required sample size is not affected much by the size of the population.

\(^1\)see page 5
1.1.3 Our Goal: Obtaining Random Samples

We are left with the problem of efficiently obtaining a random sample. For the outlined approach to be worthwhile, sampling should be faster than exhaustive traversal. We will permit the sampling algorithms to be linear in the size of the sample, but not of the population. Moreover, we will keep the dependency on population size insubstantially small, therefore outperforming exhaustive traversal on larger populations. In addition, we will strive to accomplish that without relying on auxiliary data structures.

1.1.4 Applications

Sampling can be used for probabilistic checking: Given a set of records one can get an estimate how many are faulty by checking a random sample. This is used in financial auditing, fissile materials auditing and quality control in the manufacturing industry.

Sampling is also used for selecting representative subjects for real-world statistical research, e.g. for selecting interview partners for opinion polls or market research.

A third area of application is exploratory data analysis.

Olken’s dissertation describes these - and other - applications in detail. If you are interested, go take a look ;)

1.2 Definitions

We say that a probabilistic algorithm computes a random variable $X$ iff its result is distributed like $X$. If $A$ is a set, a random element of $A$ denotes a random variable containing an element of $A$ and a uniformly random element of $A$ denotes a random variable defined by the uniform distribution over $A$.

1.2.1 Random Sample

Definition 1 (1-sample) a simple 1-sample from a set $R$ is a uniformly random element of $R$

Definition 2 (random sample with replacement) a sample with replacement of size $s$ from a set $R$ is a multiset of $s$ independent 1-samples from $R$

Definition 3 (random sample without replacement) a sample without replacement of size $s$ from a set $R$ is a random subset of size $s$

In this text I will only speak about simple random samples with replacement.

1.2.2 Classification of Algorithms

Sampling algorithms may be

- iterative
  the algorithm iteratively computes $s$ 1-samples, then returns them
• scan
  the algorithm scans $R$ maintaining a sample of the records encountered so far

Typically, scan algorithms will first decide which element to include next (skipping the intermediate elements), in which case they will not read substantially more elements than needed. However, scan algorithms make better use of caches and – when the set to be sampled from is stored on contiguous disk blocks – benefit from the increased efficiency of sequential disk access. But as scan algorithms face about the same problems as their iterative counterparts, but are substantially more tedious to analyze, interested readers are referred to Olken’s dissertation for specifics.

Olken’s dissertation compares sampling algorithms by comparing the expected number of disk pages they read (where the expectation aggregates over the random choices made by the algorithm). We adopt this notion in spite of its negligence of caching due to the lack of a source providing a more elaborate analysis.

2 Sampling from Indices

2.1 Basic Tools

2.1.1 Introductory Exercise

You are given a subset $A$ of $[0,1]^2$ by means of an inclusion test, i.e. a program that returns for every input $p \in [0,1]^2$ whether $p \in A$. How can you compute a 1-sample from $A$? Turn the page to see my solution.
Algorithm 1 (My solution) First obtain a candidate 1-sample from $[0, 1]^2$, then test whether it is in $A$. If it is, return it, otherwise, try again.

2.1.2 A/R Sampling
A slight generalization of the above approach is

Algorithm 2 (Acceptance/Rejection Sampling) is

1. obtain a candidate 1-sample (using some underlying algorithm)

2. if the candidate is accepted return it, otherwise start over

2.1.3 Effect of A/R Sampling on Sample Distribution
If we have some underlying 1-sampling algorithm at our disposal and then apply A/R Sampling to it, the new algorithm will compute a 1-sample, but with a different – and hopefully more useful – distribution (recall that a sampling algorithm is required to compute a specific distribution). To control this process, we derive the distribution imposed by A/R Sampling as a function of the distribution computed be the underlying algorithm and the acceptance test used. To keep our analysis generic, we permit the acceptance test to be probabilistic.

First, we observe that the recursive algorithm

$$AR(\text{underling, accept}) = \begin{cases} C & \text{if } \text{accept}(C) \\ S' & \text{otherwise} \end{cases}$$

is semantically equivalent to A/R Sampling with underlying algorithm underling and acceptance test accept.

If $S \overset{\text{def}}{=} AR(\text{underling, accept})$, we have

$$S = \begin{cases} C & \text{if } \text{accept}(C) \\ S' & \text{otherwise} \end{cases}$$

where $S'$ is the result of the recursive call. Obviously, it is distributed like $S$, but independent (the recursive call uses fresh randomness). Expressed in probabilities, we get:

$$P[S = a] = P[\text{accept}(C) \land C = a] + P[\neg \text{accept}(C) \land S' = a]$$

$$= \frac{P[\text{accept}(C) \mid C = a] \cdot P[C = a]}{P[\text{accept}(a)]} + \frac{P[S' = a] \cdot P[\neg \text{accept}(C)]}{P[S = a]} \cdot \frac{1 - P(\text{accept}(C))}{\lambda}$$

$$\text{def} = \lambda$$
\[ P[S = a] = \frac{P[\text{accept}(a)] \times P[C = a]}{\lambda} \]

For a deterministic choice of \text{accept}, A/R sampling therefore restricts the sampling domain to a decidable subset while retaining relative inclusion probabilities. Specifically, if \( C \) is uniformly distributed, so is \( S \).

For a probabilistic choice of \text{accept}, A/R sampling acts as a multiplicative filter for the relative inclusion probabilities. An application is the balancing of non-uniform inclusion probabilities: If \( C \) is not uniformly distributed, but we can evaluate its probability density function and choose to accept every candidate \( c \) with probability proportional\(^2\) to \( 1/P(C = c) \), the result \( S \) is uniformly distributed.

### 2.1.4 Efficiency of A/R Sampling

As the individual iterations are statistically independent, the necessary number of iterations is geometrically distributed and

\[ E[\text{iterations}] = \frac{1}{P[\text{accept}(C)]]} \]

### 2.2 Sampling from \( B^+ \) trees / Early Abort

A \( B^+ \) tree is a variant of a B-Tree where only leaf nodes store records (inner nodes store just keys, permitting a larger fan-out). Let \( n \) be the number of records in the tree, \( h \) be its height and \( k \) its minimum fan-out. Then, its maximum fan-out is \( 2^k - 1 \). When I give numbers, I will assume typical values of \( k = 128 \) and \( h = 4 \).

#### 2.2.1 Genesis

We have to obtain a uniformly random leaf. We do this by obtaining a uniformly random path and then taking its end. In a complete \( B^+ \) tree, obtaining a random path is easy: start at the root and, at every node, follow a uniformly random child until a leaf is reached.

Generalizing this approach to incomplete \( B^+ \) trees, we would like to restrict the sampling domain to existing leaves while retaining their relative inclusion probabilities. We do this by using A/R Sampling, accepting a path if it exists in the real \( B^+ \) tree, which we check by following it.

\(^2\)of course, choosing a suitable constant of proportionality requires that we know a positive lower bound for the probability density function
2.2.2 The Algorithm

Algorithm 3 (Early Abort) \textit{repeat until success:}

1. choose a uniformly random path in the complete tree of same height

2. try to follow the path in the real tree:
   - on arriving at some node, choose the same child reference the selected path did
   - if it doesn’t exist: restart (failure)
   - if it denotes an inner node: go to it
   - if it denotes a leaf: return it (success!)

2.2.3 Correctness

We have to show that Early Abort computes a uniform 1-sample. We observe that at every inner node, every choice has probability \( \frac{1}{2^k-1} \). Also, all paths have the same length and thus the same number of choices. Consequently, all paths are chosen with the same probability. A leaf is returned iff the unique path ending in it is chosen. Therefore, all leaves are selected with the same probability.

2.2.4 Efficiency

To characterize the efficiency of Early Abort, we first determine the expected number of iterations needed. In a second step, we determine the expected cost per iteration.

For the first step, let \( h \) be the height of the tree and \( n \) the number of elements it contains. To simplify our analysis, we assume that all inner nodes have \( \beta(2k-1) \) children. Then the probability of survival is \( \beta \) at each node. Early Abort reaches a leaf iff it survives \( h \) times; therefore, the probability to reach a leaf is \( \beta^{h-1} \) (a trivial optimization prevents restarting at the root). The individual iterations being independent, the number of iterations is geometrically distributed; in expectation, we therefore need \( \beta^{h+1} \) iterations.

For the second step, let \( L \) denote the path length in some iteration. Then,

\[
E[L] = \sum_{i=1}^{h-1} P(L \geq i) = \sum_{i=1}^{h-1} \beta^{i-1} = \frac{1 - \beta^{h-1}}{1 - \beta}
\]

For the total cost, we get

\[
E[D] \approx 3E[\text{attempts}] \times E[L] = \beta^{-h+1} \frac{1 - \beta^{h-1}}{1 - \beta}
\]

\(^3\text{this is not an equality, as attempts and } L \text{ are dependent. However, a more accurate calculation would show this is a good approximation.}\)
The above analysis assumed caching of the root node only. Typically, several levels of the tree will be available in main memory. It turns out only the number of uncached levels $h'$ contribute to the expected cost:

$$E[D] \approx \beta^{-h'} - \beta^{h'} \frac{1}{1-\beta}$$

Figure 1 illustrates the crucial influence of $\beta$ on performance: In the worst case of $\beta \approx \frac{1}{2}$, $E[D] \approx 2^{h'+1} - 2$, which is better than it looks:

$$n \geq k^h = 2^{(\log_2 k)h} \geq E[D] \log_2 k$$

$$E[D] \leq n^{\frac{1}{\log_2 k}}$$

which, for a typical $k = 128$, yields

$$E[D] \leq \sqrt{n}$$

So if we use Early Abort, even assuming the worst, we can grow the database without substantially affecting the cost per 1-sample, which constitutes a sharp

---

4Proof sketch: We can imagine to cut the tree above the lowest cached level, resulting in a forest of trees with cached root nodes. Then, locating a tree in the forest incurs zero cost. Also, due to our assumption of constant fan-out, all trees in that forest have the same shape and thus the same expected access cost, which can be estimated using the preceding analysis.
contrast to naive algorithms which read the entire database regardless of sample size.

In the best case $\beta = 1$, we get $E[D] = E[L] = h'$, i.e. Early Abort is as efficient as a lookup query.

It may sound bad we are only that good in a best case as intuitively, a 1-sample could also be obtained by looking up a random key. This intuition is misleading, though, as a 1-sample is not just a random element, but a \textit{uniform} random element, and performing a lookup of a uniformly random key will not achieve uniform inclusion probabilities, as records in a dense region of the key space will be included with smaller probability.

### 2.3 Sampling from Ranked Trees

#### 2.3.1 Method

In contrast to a uniform distribution on keys, a uniform distribution on ranks does induce a uniform distribution on records. To use this, we need to be able to look up a record based on its rank (and be able to find out the number of records in the tree). A data structure supporting such queries is the ranked search tree, where each reference to a subtree also stores the subtree’s cardinality, i.e. the number of elements in the subtree. A rank lookup can then be performed analogously to a key lookup, requiring $h - 1$ blocks to be fetched from disk.

#### 2.3.2 Comparison to Early Abort

The main advantage of the presented method is the guarantee of efficiency regardless of the shape of the tree or of the algorithm’s random choices. However, it relies on rank information which requires additional storage in the tree’s nodes and thus reduces $k$. In addition to taking up storage, this information must also be maintained: Every insert or delete operation modifies the cardinality counters of all ancestor nodes of the subject leaf, which requires $h - 1$ disk writes. Also, this is severely limiting in a concurrent setting, as the root node must be updated by every such operation.

Sampling from ranked trees therefore trades fast insert/delete operations for faster sampling. The former being more frequent in most applications, this will usually not be worthwhile.

It is worth noting that although ranked trees are impractical in most settings, \textit{approximately} ranked trees need not be $\beta$ while still accelerating sampling.

---

3 Sampling from Queries

3.1 Selection

3.1.1 Problem Statement

Consider a query of the form

\[
\text{sample } s \text{ from (}
\begin{align*}
&\text{select } * \\
&\text{from } R \\
&\text{where } \text{pred}
\end{align*}
\)
\]

This section details a few ways to support it. For purposes of comparison, let \( s \) be the size of the sample and

\[
\rho \overset{\text{def}}{=} \frac{|\text{select}_{\text{pred}}(R)|}{|R|}
\]

3.1.2 First Thought

A straightforward approach would be to first compute the selection (dumping it to some file) and then use file sampling\(^6\) to extract a sample.

This requires computing the entire selection regardless of sample size:

\[
E[D] \geq \rho \frac{|R| \text{recordsize}}{\text{blocksize}}
\]

Fortunately, there are alternatives:

3.1.3 Without Index

As there is no index, we have no means to efficiently locate the elements satisfying \( \text{pred} \). All we have is access to \( R \) and \( \text{pred} \) while seeking a 1-sample from the subset satisfying \( R \). Does this sound familiar\(^7\)?

Reusing the generic analysis for A/R Sampling (assuming the cost to obtain \( C \) to be independent from \( \text{pred}(C) \)) we get:

\[
E[D] = s \frac{E[\text{obtain } C]}{\rho}
\]

The cost of obtaining \( C \), and therefore the overall efficiency of this method, depends on the representation of \( R \). Assuming it is a \( B^+ \) tree, we can use Early Abort and get\(^8\):

\[
E[D] \leq \frac{s \sqrt{|R|}}{\rho}
\]

\(^6\)as files support lookup by rank (also known as random access :) ) this is trivial.

\(^7\)if not, check section 2.1

\(^8\)assuming caching of the root node only and a typical \( k = 2^7 \)
We observe that this is linear in $s$, nearly independent of the size of the relation, but very sensitive to small values of $\rho$, i.e. to highly selective predicates $\text{pred}$.

### 3.1.4 With Index

In the fortunate situation where the set of selected elements corresponds to a (union of few) subtrees of the index, we can efficiently construct a $B^+$ tree containing exactly the selected elements from the index and then sample from that tree using Early Abort.

$$E[D] = E[\text{tree construction}] + s(1 + E[\text{Early Abort}]) \leq 7\sqrt{|R|}$$

We observe that this method is for all $\rho$ about as fast as the preceding one for its best $\rho$.

If, moreover, the index is ranked, we can use the faster algorithms to sample from the tree, leading to a cost of

$$E[D] = E[\text{tree construction}] + sh$$

### 3.1.5 Summary

In essence, we have seen different kinds of algorithms to sample from a selection. The first one was linear in $|R|$. Unsatisfied with that, we considered algorithms only marginally affected by $|R|$, which require that the predicate $\text{pred}$ is not very selective or that $\text{pred}$ is simple enough and the necessary indices are available.

### 3.2 Further Work

Olken describes similar, yet more intricate, algorithms for

- projection
- union
- intersection
- difference
- join

As the results of selection, intersection and join may be substantially smaller than their arguments, computing the entire sub-query may be more efficient for these operators. Also, some of these algorithms can not yet be used compositionally, e.g. due to a dependence on the arguments’ cardinality.
4 Conclusion

We have seen that many questions about data can be answered well enough using a sufficiently large random sample, which can be obtained orders of magnitude faster than the entire dataset. We also conclude that sampling can often be performed faster inside the DBMS, as extracting the data amounts to copying the entire population, whereas internal sampling algorithms often manage to read by far less than that.