Simple Random Sampling from Relational Databases*

Frank Olken
Doron Rotem†

Computer Science Research Dept.
Lawrence Berkeley Laboratory
Berkeley, CA 94720

Abstract

Sampling is a fundamental operation for the auditing and statistical analysis of large databases. It is not well supported in existing relational database management systems. We discuss how to obtain samples from the results of relational queries without first performing the query. Specifically, we examine simple random sampling from selections, projections, joins, unions, and intersections. We discuss data structures and algorithms for sampling, and their performance. We show that samples of relational queries can often be computed for a small fraction of the effort of computing the entire relational query, i.e., in time proportional to sample size, rather than time proportional to the size of the full result of the relational query.

1 Introduction

This paper is concerned with the question of how to efficiently extract random samples of relational queries from a relational data management system. Our goal is to obtain the samples without first computing the entire query result which is to be sampled. The full paper, [OR86], begins the discussion of this topic by treating simple random sampling of hashed, grid, and B^+−tree files. Here we treat only sampling the results of individual relational operators: selection, projection, intersection, union, difference, and join.

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 reponse time or resource usage. The savings generated by sampling may be due to reductions in the cost (in reponse time or resources, CPU and I/O time) in retrieving the data from the DBMS. Retrieval costs are significant when dealing with large administrative or scientific databases.

In addition savings may result from reductions in the cost of subsequent "post processing" of the sample. Such "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, physical audits of financial records and medical examinations of sampled patients for epidemiological studies. References are given in the full paper [OR86].

Clearly for sampling to be useful, the application must not require the complete answer to the query. Thus random sampling is typically used to support statistical analysis of a dataset, either to estimate parameters of interest or for hypothesis testing. See [Coc77] for a classic treatment of the statistical methodology. Applications include scientific

*Issued as tech report LBL–20707(condensed). The full paper issued as tech report LBL–20707. This work was supported by the Director, Office of Energy Research, Office of Basic Research Sciences, Division of Engineering, Mathematical and Geosciences of the U.S. Department of Energy under Contract DE-AC03-76SF00098.

†On leave from Univ. of Waterloo, Canada. Partially supported by Canadian NSERC Grant A3055.

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 Why put sampling in DBMS?

Given that one wants to perform sampling, is it worthwhile to put the sampling operator into the DBMS?

We believe that one should put sampling operators into the DBMS for reasons of efficiency. By embedding the sampling within the query evaluation, we can reduce the amount of data which must be retrieved in order to answer sampling queries, and can exploit indices created by the DBMS.

Sampling can be used in the DBMS to provide cheap estimates of the answers of aggregate queries, [Mor80]. Sampling may also be used to estimate database parameters used by the query optimizer to choose query evaluation plans, [Wil84].

1.3 Organization of Paper

The condensed paper is organized into six sections. In Section 2 we explain some of the different types of sampling. In Section 3 we discuss the various efficiency metrics used to evaluate competing algorithms. In Section 4 we review basic sampling techniques from a single flat file which we use in this paper. In Section 5 we discuss sampling from individual relational operators. Finally, in Section 6 we state our conclusions.

In the full paper, [OR86], we also discuss the use of auxiliary "indices" to improve random access sampling of hash files, grid files, and B^+-tree files.

2 Types of Sampling

There are a variety of types of sampling which may be performed. The various types of sampling can be classified according to:

1. the manner in which the sample size is determined,
2. whether the sample is drawn with or without replacement,
3. whether access pattern is random or sequential,
4. whether or not the size of the population from which the sample is drawn is known,
5. whether or not each record has a uniform inclusion probability.

The sample inclusion probabilities for individual records may be *uniform* (an *unweighted or simple random sample (SRS)*) or they may be *weighted* according to some attribute of the record.

In this paper we will deal primarily with fixed size simple random samples. In the Section 4.1 we show how to convert between simple random samples with and without replacement. We include a short discussion of weighted sampling of an existing file, because it is used to implement simple random sampling of some relational operators. We deal with sampling from both known and unknown population sizes.

3 Efficiency Measures

We shall assume that the database resides on disk and thus measure efficiency in terms of disk blocks read.

Converting the number of records read into the number of disk blocks read is fairly well understood if the data is uniformly distributed (see [Yao77]). Christodoulakis [Chr84] has analyzed the case where the data is not uniformly distributed. For the sake of brevity and clarity we will typically assume that the fraction of records sampled from a relation is sufficiently small that each record sampled results in a disk read. Hence we will approximate the number of disk blocks read by the the number of records read. Obviously this is a gross simplification, but it can easily be corrected.

4 Basic Techniques

In this section we develop basic techniques for sampling from single files which either already exist or are being generated in their entirety. In this section we discuss conversion between simple random samples with and without replacement, weighted random sampling, and previous work on sampling from flat files.

In the full paper, [OR86], we develop techniques for sampling from some types of files which are common in DBMSs: such as files with variable numbers of records per block, hashed or grid files, and B^+-tree files.

4.1 Converting Samples

In this section we discuss how one converts between simple random samples with replacement and those without replacement.

Definition 1 *Samples without replacement are those in which each element of the sampled population appears at most once.*

For simple random sampling, a sample without replacement can be obtained from a sample with replacement by simply removing the duplicates. Of course, the sample size may thereby be reduced, so that additional elements of the population may have to be sampled.

The most efficient way to detect duplicates is usually to construct a hash table of the sample elements. This can be done in $O(s)$ time and space. Duplicate detection can be performed incrementally as the sample is collected, so that the sampling may be continued until the target sample size is reached. See [EN82].

When simple random sampling sequentially from a file (or the output of a query) it may be simpler to sample without replacement. Suppose, however, that one wants a sample with replacement. Such a SRSWR is needed when sampling from joins, as will be seen later. The SRSWOR sample can be converted to SRSWR by synthetically generating the duplicates. Essentially we generate a simple random sample with replacement from an index set of integers and then we construct a random mapping between the original sample without replacement and our sample from the index set. It is easy to see that this will produce a simple random sample with replacement (SRSWR). It requires random access only to the SRSWOR, which will usually fit in main memory, unlike the original population. Assuming that we use a hash table to check for duplicates, the conversion can be done in O(s) time and space. For details see the full paper.

4.2 Weighted Random Sampling

We will show later that, in order to obtain *simple* random samples of some types of files or relational queries, it is often necessary to compute *weighted* random samples. Hence we include a brief treatment of how to calculate a *weighted* random sample from an exisiting file with fixed blocking. Two methods of weighted sampling, acceptance/rejection sampling and partial sum trees, are compared below.

4.2.1 Acceptance/Rejection Sampling

The basic tactic used in this paper is acceptance/rejection sampling. 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 Ω 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 Ω 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. Other methods, such as the partial sum tree method discussed below, require preprocessing the entire table of weights.

4.2.2 Partial Sum Trees

Wong and Easton [WE80] proposed to use binary partial sum trees to expedite weighted sampling.

As above, consider the file of N records, in which each record r_j has inclusion probability w_j in a sample of size 1. Binary partial sum trees are simply binary trees with N leaves, each containing one record r_j and its weight w_j. Each internal node contains the sum of the weights of all the data nodes (i.e., leaves) in its subtree. Each record, r_j, can be thought to span an interval $[\sum_1^{j-1} w_j, \sum_1^j w_j)$, of length w_j.

A sample of size 1 is obtained by generating a uniform random number, u, which ranges between 0 to W, where $W = \sum_1^N w_j$. The partial sum tree is then traversed from root to leaf to identify the record which spans the location u.

The height of the tree is $O(\log N)$, where N is the number of records. Hence the time to obtain a sample of size s is $O(s \log N)$. The tree can also be updated in time $O(\log N)$ should the record weights be modified, or if sampling without replacement is desired.

Partial sum trees can be constructed in the form of B-trees, in order to minimize disk accesses by increasing the tree fanout (and hence the radix of the log). Alternatively, a partial sum tree may be embedded into a B-tree index on some domain.

Partial sum tree sampling may well outperform acceptance/rejection sampling. Essentially, it is another index, specially suited to sampling. However, it is practical only when the weights are known beforehand. Like any other index, it increases the cost of updates.

However, we believe that updates will greatly outnumber sampling queries in most applications. For this reason, and for the sake of brevity, we will discuss only acceptance/rejection methods in this paper.

4.3 A Review of Sampling from Files

In Table 1 we list the major results on sampling from a single flat file (with fixed blocking), with citations to the relevant algorithms. We employ some of these techniques in our work on query sampling. Also see [Dev86].

5 Sampling from Relational Operators

In this section we show how to sample the output of individual relational operators such as selection, projection, intersection, union, difference, and join. These sampling techniques form the basic building blocks for sampling from more complex composite queries. The techniques entail a synthesis of the basic file sampling techniques and algorithms for implementing relational operators. We discuss only simple random sampling.

In order to facilitate the exposition, we treat the simpler relational operators first, leaving the most interesting results concerning joins for last.

Our cost measure is the number of disk pages read, denoted as D. Usually we will be interested in the expected number of disk pages read, $E(D)$.

5.1 Notation

The sampling operator will be denoted as ψ. Sampling method and size will be denoted by subscripts. Except as noted simple random sampling without replacement (SRSWOR) is the default sampling method, e.g., $\psi_{100}(R)$ or $\psi_{SRSWOR,500}(S)$.

More complex sampling schemes will be described via the iteration operators:
WR(s,< *expr* >), and WOR(s,< *expr* >), which indicate that < *expr* > (a sampling expression) is to be repeatedly evaluated until a sample of size s obtained (with or without replacement respectively).

Definition 2 *Two sampling schemes, $A(R)$ and $B(R)$ of relation R are said to be equivalent, denoted by $A \Leftrightarrow B$, if, for every possible instance r of relation R they generate the same size samples, and the inclusion probability for each element of the population is the same in both schemes. Note that the samples are not necessarily identical.*

Definition 3 $MIX(\alpha, < expr_1 >, < expr_2 >)$ *denotes a random mixture of two sampling schemes. It indicates that we sample according to $< expr_1 >$ with probability α, and with probability $1 - \alpha$ we sample according to $< expr_2 >$.*

MIX is used to implement sampling from unions.

Definition 4 $ACCEPT(\alpha, < expr >)$ *indicates that we accept the sample element generated according to $< expr >$ with probability α.*

$ACCEPT$ is used to implement sampling from projections and joins.

5.2 Selection

We denote the selection of records satisfying predicate *pred* from relation R by $\sigma_{pred}(R)$ The number of records in relation R is n. The fraction of records of relation A which satisfies predicate *pred* is ρ_{pred}. Hence $n\rho_{pred}$ is the number of records in relation R which satisfy the selection predicate.

Selection is unique in that it correctly commutes with the sampling operator, i.e., selecting from a simple random sample generates a simple random sample of a selection.

Theorem 1

$$\psi_s(\sigma_{pred}(R)) \Leftrightarrow WOR(s, \sigma_{pred}(\psi_1(R)))$$

Proof: For records which do not satisfy the predicate, the inclusion probability is obviously zero on both sides.

In the sampling scheme on the lefthand side the inclusion probability, p, for any record r which satisfies the selection predicate is:

$$p = s/(n\rho_{pred}) \qquad (2)$$

i.e., all such records have equal inclusion probabilities.

On the righthand side, we repeatedly sample one record from R, evaluate the selection predicate, and then retain it if it satisfies the selection predicate and is not a duplicate. This continues until we have a sample size s.

Since the selection operator does not alter the inclusion probabilities of those records which satisfy the selection predicate, they remain equi-probable. From

Type of sampling	Citation	Expected Disk Accesses
Simple Random Sampling with replacement		$O(s)$
Simple Random Sampling with replacement with variable blocking	[OR86]	$O(s(b_{max}/b_{avg}))$
Simple Random Sampling without replacement	[EN82]	$O(s)$
Weighted Random Sampling	[WE80]	$O(s \log n)$
Sequential Random Sampling, known population size	[FMR62]	$O(n/b_{avg})$
	[Vit84]	$O(s)$
Sequential Random Sampling, unknown population size	[Vit85]	$O(n/b_{avg})$
	[Vit85]	$O(s(1+\log(n/s))$

Table 1: Basic Sampling Techniques from a single file

Note: s = sample size, n = population size, b_{max} = maximum number of records in a block, b_{avg} = average number of records in a block. 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.

the definition of the *WOR* iterator, we are assured that the sample size is s distinct records. □

Techniques for sampling from selections may be classifed according as to whether they use an index, or scan the entire relation. The first class can be further classified according to whether the index contains rank information, which permits random access to the $j'th$ ranked record. We shall assume that the index is a single attribute record-level index constructed as a B^+-tree as discussed in the full paper, [OR86]. Except as noted, we assume that the predicate can be fully resolved by the index. Based on this classification schema, we have the following algorithms:

- KSKIPI: sample sequentially via random access SKIPs in Index,
- RAI: Random Access sample via Index until desired sample size is obtained,
- SCANI: sample sequentially via Index SCANning every relevant index page,
- RA: Random Access sample directly until desired sample size is obtained,
- SCAN: sample sequentially SCANning every page of relation,

In order to generate random accesses via the index, we must assume that the index includes rank information as discussed in Section 5.3.

The first method, sequential sampling via random access skips (KSKIPI) can be expedited [Vit84] if the population size (number of tuples which qualify on the predicate) is known, i.e., computable from the rank information in the index. In this case the expected number of disk accesses is given by:

$$E(D_{KSKIPI}) \approx (s(1+\log_f(\frac{n\rho_{pred}}{sf}))) \qquad (3)$$

Here f is the average fan-out of each node in the B^+-tree index. The log term is due to average height in the tree we must backtrack for each skip. We assume one additional access for each element of the sample to actually retrieve the sampled record.

Again assuming rank information in the index, the second method, random probes of the subtree of the index selected by the predicate (RAI), has an expected cost of:

$$E(D_{RAI}) \approx (s(1+\log_f(\frac{n\rho_{pred}}{f}))). \qquad (4)$$

Clearly, KSKIPI is always more efficient than RAI for simple predicates. However, there may be occasions in which multi-attribute predicates are specified for which only a single index is available. This precludes the use of KSKIPI, because we don't know the size or the identity of the population satisfying the multi-attribute predicate. However, we can continue to use RAI on one index, and evaluate the multi-attribute predicate on each record sampled.

The third method, sequentially sampling via the index consists of finding the pages of the index which point to records which satisfy the predicate, and then sequentially scanning and sampling each such index page, assuming that successive index pages are chained. The sequential sampling would be done with a reservoir method such as [Vit85], which does not require a known population size. This method would be used when the index does not contain the rank information needed for RAI or KSKIPI. It has an expected cost of:

$$E(D_{SCANI}) \approx \log_f(\frac{n}{f}) + \frac{n\rho_{pred}}{f} + s \qquad (5)$$

The fourth method, direct random access sampling (RA), does not require any index. For a relation with a fixed blocking factor the number disk accesses required to obtain s distinct records is a negative hypergeometric distribution whose mean is approximately given by:

$$E(D_{RA}) \approx s/\rho_{pred} \qquad (6)$$

assuming that $s \ll n\rho_{pred}$. The advantage of this method is that it does not require an index. If ρ_{pred} is close to 1 this method avoids superfluous accesses to the index. If ρ_{pred} is very small the SCAN method is to be preferred.

The fifth method, SCAN, consists of simply scanning the entire relation to perform the selection, with a pipelined sequential sampling of the result. The number of page accesses is simply the size of the relation:

$$E(D_{SCAN}) = n/b_R \qquad (7)$$

Here b_R is the blocking factor for relation R.

5.3 Projection

For simplicity we only consider projection on a single domain. Similar results hold for projection on multiple domains. Simlarly, for expository purpose we treat only sampling with replacement. As shown earlier extensions to sampling without replacement are straightforward. We denote the projection of relation R onto domain A as $\pi_A(R)$.

Let A be an attribute defined on the domain $a_1, a_2, ..., a_m$. The set $R.a_i$ includes all the tuples in R with value a_i on the attribute A.

Definition 5 *The minimum frequency of the attribute A in relation R, denoted as $|R.a|_{min}$, is the minimum cardinality in relation R of any projection domain value a_i:*

Theorem 2 *If A is not a key of R then*

$$\psi_s(\pi_A(R)) \not\Leftrightarrow \pi_A(\psi_s(R))$$

Proof: A counterexample is given in the full paper [OR86].

Projection does not generally commute with sampling because the projection operator removes duplicates. Hence, interchanging projection and sampling will produce uneven inclusion probabilities. □

However, if the attribute A is a key of the relation R, then there will be no duplicate values of A in R, hence projection and sampling can be exchanged with impunity.

Theorem 3

$$\psi_{SRSWR,s}(\pi_A(R)) \Leftrightarrow$$

$$\pi_A(WR(s, ACCEPT(\frac{|(R.a)|_{min}}{|(R.a_i)|}, \psi_{SRSWR,1}(R))))$$

Proof: On the righthand side we sample with replacement from relation R first. Hence, each value a_i in the projection domain A would have an inclusion probability of $|(R.a_i)|/|R|$. Since we want uniform inclusion probabilities on the projected domain values, we employ acceptance/rejection sampling to correct the inclusion probabilities. The acceptance probability for a tuple with value a_i in the projection domain A is given as:

$$p_i = \frac{|(R.a)|_{min}}{|(R.a_i)|} \qquad (8)$$

Hence,for each iteration the inclusion probability for each distinct a_i is:

$$p = \frac{|(R.a)|_{min}}{|R|} \qquad (9)$$

We repeat this until we have s distinct records in our sample. □

Hence the expected cost is:

$$E(D) \approx s\frac{|(R.a)|_{avg}}{|(R.a)|_{min}} \qquad (10)$$

assuming $s \ll |\pi_A(R)|$, where $|(R.a)|_{avg} = |R|/m$ is the average cardinality of attribute A over all attribute values a_i present in the relation. Here we have assumed that relation R is hashed on the projection domain so that records may be retrieved in a single access.

In order for the above algorithm to work we must be able to readily determine the cardinality (number of duplicates) of each projected tuple. This requires that the relation to be projected must be either sorted, indexed or hashed on the projection domain. Also we must either know $|(R.a)|_{min}$ or replace it with a lower bound of 1, at the expense of reduced efficiency.

The case in which the relation to be projected is not "indexed" is discussed in the full paper, [OR86].

5.4 Intersection

We denote the intersection of two distinct relations R and T as $R \cap T$.

While it is possible to distribute sampling over intersection and still preserve uniform inclusion probabilities, the resulting computation is so inefficient that it is rarely worthwhile.

Theorem 4

$$\psi_s(R \cap T) \Leftrightarrow WOR(s, \psi_1(R) \cap T) \quad (11)$$
$$\Leftrightarrow WOR(s, R \cap \psi_1(T)) \quad (12)$$

Proof: Consider the first case. From the lefthand side we have the inclusion proability for tuples in $R \cap T$ is $s/|R \cap T|$, zero otherwise. For the righthand side we have the inclusion probability for all tuples in the intersection of R and T is $s/|R \cap T|$, zero otherwise. In each case we have a simple random sample. □

We thus have our choice of which relation to sample from and which relation to do the intersection with. Typically, if only relation R has an index, then we would sample from T and then intersect with R using its index, since the alternative would require scanning all of T in order to perform the intersection.

If both R and T have indices we must consider the relative costs of the two options based on the size of the relations, the type of index (hash, B^+−tree, primary or secondary), and the blocking factors for each relation.

If neither R nor T have indices, then we would sample from the larger relation, so that the intersection scan can be performed on the smaller relation.

Definition 6 *Define the intersection selectivities* ρ_R, ρ_T *as:*

$$\rho_R = |R \cap T|/|R|, \quad \rho_T = |R \cap T|/|T| \quad (13)$$

Then the cost in disk accesses of sampling from R and then checking for inclusion in T, assuming T has a B^+−tree index is:

$$E(D_R) \approx \frac{s}{\rho_R}(1 + \log_{f_{TI}}(\frac{|T|}{f_{TI}})) \quad (14)$$

where f_TI is the the average fan-out of the B^+−tree index to relation T. Again we assume that $s \ll |R \cap T|$, i.e., we neglect the extra cost of sampling without replacement.

An analogous formula for $E(D_T)$ can be written if we sample from relation T and check the intersection in relation R. The choice of which file to sample from can be made by comparing the values of the two cost formulas.

5.5 Difference

We denote the difference of two relations R and T as $R - T$.

Theorem 5

$$For\ all\ k : \psi_k(R - T) \not\Leftrightarrow \psi_k(R) - \psi_k(T)$$

Proof: Interchanging sampling and difference fails because elements in $R \cap T$ but not in $\psi_k(T)$ may be erroneously included in $\psi_k(R) - \psi_k(T)$. □

Theorem 6

$$\psi_s(R - T) \Leftrightarrow WOR(s, \psi_1(R) - T)$$

i.e., sampling from the difference of two relations is equivalent to sampling from the first relation and then taking the difference.

Proof: Clearly the sampling scheme on the righthand size will produce a sample without replacement of the desired size. It remains to be shown that the sample is from $R - T$ and that each element in $R - T$ has an equal inclusion probability. Since $\psi_1(R) \in R$ it follows that $\psi_1(R) - T \in (R - T)$. Since $\psi_1(R)$ has uniform inclusion probabilities over all elements of R and set differencing with T does not alter the inclusion probabilities of records in $R - T$, it follows that the righthand side sampling scheme has uniform inclusion probabilities for records in $R - T$. □

Thus sampling from relation differences is very similar to sampling from relation intersections. We sample from R and then check that the tuple is not in T. Hence the expected cost assuming T has a B^+−tree index is approximately:

$$E(D) \approx \frac{s}{(1 - \rho_R)}(1 + \log_{f_{TI}}(\frac{|T|}{f_{TI}})) \quad (15)$$

Again we assume that $s \ll |R - T|$, i.e., we neglect the extra cost of sampling without replacement.

5.6 Union

We denote the union of two distinct relations R and T as $R \cup T$.

Theorem 7

$$For\ any\ S_1, S_2 : \psi_s(R \cup T) \not\Leftrightarrow \psi_{S_1}(R) \cup \psi_{S_2}(T) \quad (16)$$

Proof: Interchanging sampling and union fails because all elements on lefthand side have identical inclusion probabilities of $s/|R \cup T|$, whereas the righthand side inclusion probabilities for elements in the

intersection $R \cap T$ are $\frac{S_1}{|R|} + \frac{S_2}{|T|} - \frac{S_1 S_2}{|R||T|}$ whereas the inclusion probability for elements in $R - T$ is $S_1/|R|$ and the inclusion probability for elements in $T - R$ is $S_2/|T|$. Hence elements in $R \cap T$ do not have same inclusion probability as elements in $(R \cup T) - (R \cap T)$. □

The correct treatment of sampling from unions requires that we sample elements of intersection only once. Observe that:

$$R \cup T = R \cup (T - R) \tag{17}$$

Theorem 8

$$\psi(R \cup T) \Leftrightarrow$$

$$WOR(s, MIX(\frac{|R|}{|R| + |T|}, \psi_1(R), (\psi_1(T) - R))$$

Recall that $MIX(\alpha, < expr_1 >, < expr_2 >)$ indicates that we sample according to $< expr_1 >$ with probability α, and with probability $1 - \alpha$ we sample according to $< expr_2 >$.

Proof: Omitted. □

Then to generate the a single sample of $R \cup T$ we repeat the following algorithm until a sample is accepted:

```
begin
i := RAND(1, |R| + |T|);
If i ≤ |R|
   then get record i from R.
   else
        begin
              j := i − |R|;
              Get record j from T.
              Check if record j is in R.
              If so, discard record j,
              otherwise retain it.
        end
endif
end
```

Assuming B^+−tree indices, and $s \ll |R \cup T|$, we have the expected number of iterations of the above algorithm to obtain a single sample is:

$$E(l_R) = \frac{(|R| + |T|)}{(|R \cup T|)} \tag{18}$$

For each iteration, we sample T (at a cost of one disk access), and the check the B^+−tree index to R. Thus each iteration has a cost of:

$$(1 + \frac{|T|}{|R| + |T|} \log_{f_{RI}}(\frac{|R|}{f_{RI}})) \tag{19}$$

5.7 Join

Given two relations R and T, let the relation W be the result of their equijoin, i.e., $W = R \underset{R.x=T.x}{\bowtie} T$. In this section we describe algorithms for sampling from W. For reasons of efficiency we wish to avoid computing the full join. For simplicity of exposition we discuss only sampling with replacement in this section. As shown earlier, conversion to sampling without replacement is straightforward.

Sampling from W can be done in different ways depending on the initial structure of the relations R and S. Some important factors in determining the sampling method are:

1. Is the join attribute a key in one or more of the joined relations?
2. Are the relations R or T indexed or hashed on the join attribute ?
3. Is the join selectivity factor large?

In this section we will cover some of the basic methods and evaluate them with respect to their efficiency. First a few notations and definitions.

We denote the semi-join of relation R with relation S over domains A of R and domain B of S as $R \underset{A=B}{\ltimes} S$. Let X be an attribute defined on the domain $x_1, x_2, ..., x_m$. The set $R.x_i$ includes all the tuples in R with value x_i on the attribute X. The join selectivity factor $\rho(R \underset{R.x=T.x}{\bowtie} T)$ of relations R and T over the attribute X is defined as

$$\rho(R \underset{R.x=T.x}{\bowtie} T) = \frac{\sum_{i=1}^{m} |R.x_i||T.x_i|}{|R||T|} \tag{20}$$

where m is the number of distinct values of the join domain. When the context is clear we will simply denote this by ρ.

5.7.1 Join without keys

First, we will deal with the case that the join attribute X is not a key in any of the relations R or T. We assume that relation T is "indexed" on the join attribute X, and that the modal frequency of the attribute X in relation T, as defined below, is also known.

Definition 7 *The modal frequency of the attribute X in relation T, denoted as $|T.x|_{max}$, is the maximum cardinality in relation T of any join domain value x_i, i.e.,*

$$|T.x|_{max} = \max_{all\ i} |T.x_i| \tag{21}$$

Theorem 9 *The following acceptance/rejection algorithm will generate a simple random sample of size s.*

$$\psi_{SRSWR,s}(R \underset{R.x=T.x}{\bowtie} T) \Leftrightarrow$$

$$WR(s, ACCEPT(\frac{|T.x_i|}{|T.x|_{max}}, \psi_1(\psi_1(R) \underset{R.x=T.x}{\bowtie} T)))$$

where $|T.x_i|$ denotes the cardinality in relation T of join domain value x_i resulting from the sample $\psi_1(R)$.

Proof: Clearly the righthand side sampling scheme will produce a sample of the requisite size from $R \underset{R.x=T.x}{\bowtie} T$. What we must show is that it will have uniform inclusion probabilities.

Each iteration of the above algorithm begins by sampling a tuple from R. Each tuple in R has inclusion probability $|R|^{-1}$. The sampled tuple, $\psi_1(R)$, is then joined with T and a random sample of the result taken, denoted $\psi_1(\psi_1(R \underset{R.x=T.x}{\bowtie} T))$. Clearly each member of $(\psi_1(R) \underset{R.x=T.x}{\bowtie} T)$ has the inclusion probability $(|T.x_i|)|)^{-1}$ as defined above. This single sample is then accepted with probability $\frac{|T.x_i|}{|T.x|_{max}}$. Hence inclusion probability of any member of $R \underset{R.x=T.x}{\bowtie} T$ is given for a single iteration by the product:

$$p = |R|^{-1}(|T.x_i|)^{-1}\frac{|T.x_i|}{|T.x|_{max}} \quad (22)$$

$$= \frac{1}{|R||T.x|_{max}} \quad (23)$$

i.e., we have uniform inclusion probabilities for each iteration. By induction, this is true for the full algorithm. □

In practice we use the equivalent following algorithm:

Algorithm - $RAJOIN_R$

comment This version of the algorithm samples R first;
For $j := 1$ **to** s **do**
set *accept* to *false*
While *accept* = *false*
begin
Choose a random record r from R.
Assume $r[X] = x_i$.
Find the cardinality of $T.x_i$.
Accept r into the sample
with probability
$p = \frac{|T.x|_i}{|T.x|_{max}}$
In case record r is accepted,
choose randomly a tuple t
of $T.x_i$ and join r with it
Store the result $r \underset{R.x=T.x}{\bowtie} t$
in the sample file.
set *accept* to *true*.
end
Endwhile
Endfor

Note that we do not actually construct the full $\psi_1(R) \underset{R.x=T.x}{\bowtie} T$, since we only need a sample of size 1 from it.

The efficiency of this method is established in the following lemma.

Lemma 1 *The expected number of times that the while loop in $RAJOIN_R$ will be performed until a sample is accepted, $E(l_R)$, is:*

$$E(l_R) = \frac{|T.x|_{max}}{\rho|T|} \quad (24)$$

Proof: The proof is given in the full paper, [OR86]. □

Hence the total efficiency (disk accesses) of the algorithm is:

$$E(D_{RAJOIN_R}) \approx sE(l_R)(1 + \log_{f_{TI}}(\frac{|T|}{f_{TI}})) + s \quad (25)$$

$$\approx s\frac{|T.x|_{max}}{\rho|T|}(1 + \log_{f_{TI}}(\frac{|T|}{f_{TI}})) + s \quad (26)$$

where f_{TI} is the average fan-out for the B^+−tree index for T. Here the log factor is the time to search the B^+−tree index of T for each sample. The last s term in each equation represents the cost of finally retrieving the sampled records from T.

If there is an index on R then an analogous algorithm $RAJOIN_T$ can be constructed by simply exchanging the roles of R and T in the above algorithm and cost analysis. If both R and T are indexed, either algorithm could be used. If the join selectivity, ρ, is very small, neither algorithm is recommended. Instead, it may be preferable to compute the full join and then sample sequentially the output of the join using [Vit85] as it is generated.

5.7.2 Join with key

Suppose that the join domain X is a key of relation T and that relation T is indexed on X. In this case, the acceptance/rejection sampling is unecessary, if we first sample from relation R. See the full paper [OR86] for details and proof.

6 Conclusions

In this paper we have begun to explore how to integrate random sampling into relational data management systems.

The contributions of this paper include: sampling techniques for relational operators, asymptotic approximate estimates of the cost (in disk accesses) of the various sampling algorithms, a concise notation for describing sampling algorithms.

Acceptance/rejection sampling comprises the basic technique by which we compensate for the effects of relational operators on inclusion probabilities. Assuming that suitable indices or access methods are available, integrating sampling into the query evaluation offers potentially major savings in query processing time.

Acknowledgements

The authors would like to thank Arie Shoshani, and Harry Wong for their encouragement and comments. Sakti Ghosh, Setrag Khoshafian, Ronnie Hibshoosh and Dan Willard also provided encouragement. Eugene Wong provided useful comments on an early oral version of the paper. Neal Rowe's pessimistic assessment of the feasibility of database sampling was a spur to the writing of this paper. Jack Morgenstein first introduced us to the idea.

References

[Chr84] Stavros Christodoulakis. Implications of certain assumptions database performance evaluation. *ACM Transactions on Database Systems*, 9(2):163–186, June 1984.

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

[Dev86] Luc Devroye. *Non-uniform Random Variate Generation.* Springer-Verlag, 1986.

[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.

[Mor80] Jacob Morgenstein. *Computer Based Management Information Systems Embodying Answer Accuracy as a User Parameter.* PhD thesis, Univ. of California, Berkeley, December 1980.

[OR86] Frank Olken and Doron Rotem. *Simple Random Sampling from Relational Databases.* Technical Report LBL-20707, Lawrence Berkeley Lab, February 1986.

[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.

[Wil84] Dan Willard. Sampling algorithms for differential batch retrieval problems (extended abstract). In *Proceedings ICALP-84*, Springer-Verlag, 1984.

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