Congressional Samples for Approximate Answering of Group-By Queries

Swarup Acharya **Phillip B. Gibbons** **Viswanath Poosala**

Information Sciences Research Center
Bell Laboratories
600 Mountain Avenue
Murray Hill NJ 07974
{swarup,gibbons,poosala}@research.bell-labs.com

Abstract

In large data warehousing environments, it is often advantageous to provide fast, approximate answers to complex decision support queries using precomputed summary statistics, such as samples. Decision support queries routinely segment the data into groups and then aggregate the information in each group (*group-by* queries). Depending on the data, there can be a wide disparity between the number of data items in each group. As a result, approximate answers based on uniform random samples of the data can result in poor accuracy for groups with very few data items, since such groups will be represented in the sample by very few (often zero) tuples.

In this paper, we propose a general class of techniques for obtaining fast, highly-accurate answers for group-by queries. These techniques rely on precomputed non-uniform (biased) samples of the data. In particular, we propose *congressional samples*, a hybrid union of uniform and biased samples. Given a fixed amount of space, congressional samples seek to maximize the accuracy for all possible group-by queries on a set of columns. We present a one pass algorithm for constructing a congressional sample and use this technique to also incrementally maintain the sample up-to-date without accessing the base relation. We also evaluate query rewriting strategies for providing approximate answers from congressional samples. Finally, we conduct an extensive set of experiments on the TPC-D database, which demonstrates the efficacy of the techniques proposed.

1 Introduction

The last few years have seen a tremendous growth in the popularity of decision support applications using large-scale databases. These applications, also known as *on-line analytical processing* (OLAP) applications, analyze historical data in a data warehouse to identify trends that can be exploited in defining new business strategies. Often, this process involves posing several complex queries over a massive database.[1] As a result, these queries can take minutes, and sometimes hours, to execute using even the state-of-the-art in data warehousing and OLAP technology.

A novel approach to address this problem, which has been receiving attention lately, is to provide *approximate answers* to the queries very quickly [HHW97, AGPR99, VW99, IP99]. This approach is particularly attractive for large-scale and exploratory applications such as OLAP. For example, a typical decision making process involves posing several preliminary queries to identify interesting regions of the data. For these queries, precise answers are often not essential. Similarly, for queries returning numerical results, the full precision of an exact answer may be overkill — the user may welcome an answer with just a few significant digits (e.g., the leading few digits of a total in the millions) if it is produced much faster. These approximate query answering systems give fast responses by running the queries on some form of summary statistics of the database, such as samples, wavelets and histograms. Additionally, the approximate answers are often supplemented with a statistical error bound to indicate the quality of the approximation to the user.[2] Because these statistics are typically much smaller in size, the query is processed very quickly. The statistics may either be generated on-the-fly after the query is posed, as in the *Online Aggregation* approach [HHW97], or may be precomputed *a priori*, as in the Aqua system [AGPR99] we have developed.

A popular technique for summarizing data is taking *samples* of the original data. In fact, this is the fundamental technique used by both the above-mentioned approaches to approximate query answering. In particular, *uniform random sampling*, in which every item in the original data set has the same probability of being sampled, is used because it mirrors the original data distribution. Also, by increasing the sample size, the system can provide more accurate responses to the user. Due to the usefulness of uniform samples, commercial DBMSs such as Oracle 8i are already supporting operators to collect uniform samples.

1.1 Limitations of Uniform Sampling

While uniform random samples provide highly-accurate answers for many classes of queries, there are important classes of queries for which they are less effective. This includes one of the most commonly occurring scenarios in

ACM SIGMOD 2000 5/00 Dallas, TX, USA
© 2000 ACM 1-58113-218-2/00/0005...$5.00

[1] A survey by the Data Warehousing Institute indicates that the average warehouse size is expected to exceed 400GB in the year 2000 and that a single decision process may involve more than ten fairly complex queries.

[2] In our discussion, *user* refers to the end-user analyzing the data in the warehouse.

decision support applications is to segment the data into groups and derive some aggregate information for these groups. This is typically done in SQL using the *group by* operation and hence we refer to them as group-by queries. For example, a group-by query on the U.S. census database containing information about every individual in the nation could be used to determine the per capita income *per state.* Often, there can be a huge discrepancy in the sizes of different groups, e.g., the state of California has nearly 70 times the population of Wyoming. As a result, a uniform random sample of the relation will contain disproportionately fewer tuples from the smaller groups (states), which leads to poor accuracy for answers on those groups because accuracy is highly dependent on the number of sample tuples that belong to that group [HHW97, AGPR99].[3] This behavior often renders the answer essentially useless to the analyst, who is interested in reliable answers for *all* groups. For example, a marketing analyst using the Census database to identify *all* states with per capita incomes above some value will not find the answer useful if the aggregates for some of the states are highly erroneous.

In fact, the inability of uniform random samples to provide accurate group-by results is a symptom of a more general problem with uniform random samples: *they are most appropriate only when the utility of the data to the user mirrors the data distribution.* Thus, when the utility of a subset of the data to the user is significantly higher relative to its size, the accuracy of the answer may not meet the user's expectation. The group-by query is one such case where a smaller group is often as important to the user as the larger groups, even though it is under-represented in the data. A multi-table query is another example: a small subset of the data in a table may dominate the query result if it joins with many tuples in other tables [AGPR99, CMN99, HH99]. The flip side of this scenario is when different logical parts of the data have equal representation, but their utility to the user is skewed. This occurs, for example, in most data warehouses where the usefulness of data degrades with time. For example, consider a business warehouse application analyzing the transactional data in the warehouse to evaluate a market for a new line of products. In this case, data from the previous year is far more important than outdated data from a decade ago. Moreover, the user is likely to ask more finer-grained queries over the more recent data. This, in turn, means that the approximate answering system has to collect more samples from the recent data, which is not achieved with a uniform random sample over the entire warehouse.[4]

To address these inadequacies of uniform random samples, we consider non-uniform (i.e., *biased*) samples in this paper, which are discussed next.

1.2 Biased Sampling for Group-by Queries

In this paper, we propose a general class of techniques for obtaining fast, highly-accurate answers for group-by queries using (precomputed) biased samples of the data. We focus on group-by queries because they are among the most important class of queries in OLAP, forming an essential part of the common *drill-down* and *roll-up* processes [Kim96, CD97]. For example, of the 22 queries in Version 2.0 of the TPC-D benchmark [TPC99], 15 are group-by queries. Our solutions, however, are more general and can be applied to a much broader set of problems wherever the limitations of uniform random samples become critical. Briefly, our techniques involve taking group-sizes into consideration while sampling, in order to provide highly-accurate answers to queries with arbitrary group-by operations (even none) and varying group-sizes. Our solutions apply and extend known techniques for subpopulation/domain/species sampling [Coc77] to the approximate answering of group-by queries. Our key extensions include considering combinations of group-by columns, construction and incremental maintenance, query rewriting, and optimizing over a query mix.

There are a number of factors affecting the quality of an answer computed from a sample, including the query, the data distribution, and the sample size. Of these, sample size is the most universal in improving answer quality across a wide range of queries and data distributions. Thus we focus in this paper on ensuring that all groups are well-represented in the sample. We consider single table queries; however, our techniques can be immediately extended to queries with foreign key joins, the most common type of joins (e.g., all joins in the TPC-D benchmark are on foreign keys), using the techniques in [AGPR99].

The techniques in this paper are tailored to *precomputed* or *materialized* samples, such as used in Aqua (see Section 2). Advantages of precomputing over sampling at query time include (1) queries can be answered quickly without accessing the original data at query time, (2) sampled tuples can be stored compactly in a few disk blocks, avoiding the overheads of random scanning, (3) no changes are needed to the DBMS's query processor and optimizer, and (4) data outliers such as small groups can be detected and incorporated into the sample. On the other hand, precomputed samples must commit to the sample before seeing the query, and are not well suited to supporting user-controlled progressive refinement [HHW97].

Our contributions are as follows:

- We introduce a hybrid union of biased and uniform samples called *congressional samples*[5], which provide statistically unbiased answers to queries with arbitrary group-by (including no group-bys), with significantly higher accuracy guarantees than uniform samples. Given a fixed amount of space, congressional samples seek to maximize the accuracy for all possible group-by queries on a set of columns. We also propose efficient strategies for executing queries on these samples.
- We develop a one pass algorithm for constructing a congressional sample without a priori knowledge of the data distribution. We use this technique to also incrementally maintain the sample as new data is inserted into the database, without accessing the base relation. This ensures that queries continue to be answered well even as the new data changes the database significantly.
- We show how congressional samples can be specialized to specific subsets of group-by queries. We also extend them to use detailed information about the data, such as variance, and to improve the answers for non-group-by queries.

[3] Based on this observation, the Online Aggregation approach employs an index striding technique to sample smaller groups at a higher rate [HHW97].

[4] Note that other common summary statistics such as histograms and wavelets suffer from this same general problem.

[5] As discussed in Section 4, the name *congressional* samples reflects an analogy to the U.S. Congress, which combines biased representation (two Senators per state, regardless of population) with more uniform representation (Representatives in proportion to a state's population).

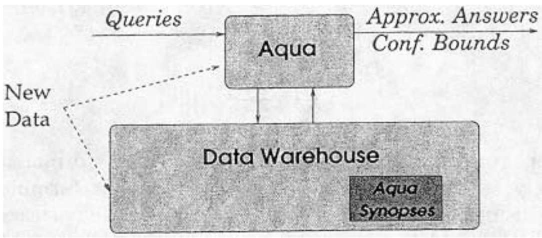

Figure 1: The Aqua architecture.

```
select l_returnflag, l_linestatus, sum(l_quantity)
from lineitem
where l_shipdate <= '01-SEP-98'
group by l_returnflag, l_linestatus;
```

(a) Original query

```
select l_returnflag, l_linestatus, 100*sum(l_quantity),
    sum_error(l_quantity) as error1
from bs_lineitem
where l_shipdate <= '01-SEP-98'
group by l_returnflag, l_linestatus;
```

(b) Rewritten query

Figure 2: Query rewriting in Aqua.

- We conduct an extensive set of experiments to establish the accuracy of congressional samples and identify an efficient execution strategy for running queries on them.

Map. The rest of this paper is as follows. In the next section, we describe Aqua, a system framework for approximate query answering. Then, we formulate the problem being addressed in this paper and in Section 4, we propose our novel sampling solutions. In Section 5, we highlight some implementation issues in using these new solutions in practice. Then, in Section 6 we propose efficient construction and maintenance techniques. The experimental study is in Section 7. In Section 8, we describe extensions of congressional samples that improve their accuracy for certain classes of queries. In Section 9, we present related work and in Section 10 we summarize the conclusions from this work.

2 Aqua System

This work is being performed as part of our efforts to enhance Aqua, an efficient decision support system providing approximate answers to queries [AGPR99, AGP99a]. Aqua maintains smaller-sized statistical summaries of the data, called *synopses*, and uses them to answer queries. A key feature of Aqua is that the system provides probabilistic error/confidence bounds on the answer, based on the Hoeffding and Chebyshev formulas [AGPR99]. Currently, the system handles arbitrarily complex SQL queries applying aggregate operations (`avg, sum, count, etc.`) over the data in the warehouse.

The high-level architecture of the Aqua system is shown in Figure 1. It is designed as a middleware software tool that can sit atop any commercial DBMS managing a data warehouse that supports ODBC connectivity. Initially, Aqua takes as an input from the warehouse administrator the space available for synopses and if available, hints on important query and data characteristics.[6] This information is then used to precompute a suitable set of synopses on the data, which are stored as regular relations in the DBMS. These synopses are also incrementally maintained up-to-date to reflect changes in the warehouse data.

l_returnflag	l_linestatus	sum(l_quantity)
A	F	3773034
N	F	100245
N	O	7459912
R	F	3779140

Figure 3: Exact answer.

l_returnflag	l_linestatus	sum(l_quantity)	error1
A	F	3.778e+06	1.4e+04
N	F	1.194e+05	2.6e+04
N	O	7.457e+06	1.9e+04
R	F	3.782e+06	1.4e+04

Figure 4: Approximate answer.

When the user poses an SQL query to the full database, Aqua rewrites the query to use the Aqua synopsis relations. The rewriting involves appropriately scaling expressions in the query, and adding further expressions to the `select` clause to compute the error bounds. An example of a simple query rewrite is shown in Figure 2. The original query is a simplified version of Query 1 of the TPC-D benchmark. The synopsis relation `bs_lineitem` is a 1% uniform random sample of the `lineitem` relation and for simplicity, the error formula for the *sum* aggregate is encapsulated in the `sum_error` function. The rewritten query is executed by the DBMS, and the results are returned to the user. The exact answer is given in Figure 3. Figure 4 shows the approximate answer and error bound provided by Aqua when using this synopsis relation, and indicates that the given approximate answer is within `error1` of the exact answer with 90% confidence[7]. The approximate answer for `l_returnflag = N` and `l_linestatus = F` is considerably worse than for the other combinations; this is the smallest group (a factor of 35 or more smaller than the others in the TPC-D database), and hence it contributes very few tuples to the sample `bs_lineitem`. This demonstrates a limitation of uniform random samples and motivates the need for the techniques proposed in Section 4.

To address the well-known problem of joins over samples [AGPR99, CMN99], Aqua collects special forms of samples, called *join synopses*, which can be viewed as uniform random samples on the results of all the interesting joins in the warehouse. In [AGPR99], we showed that join synopses are particularly effective on the *star and snowflake schemas* which are common in data warehousing [Sch97]. An interesting outcome of join synopses is that any join query involving multiple tables on the warehouse can be conceptually rewritten as a query on a *single join synopsis relation*. Due to this reason, in this paper, we restrict our discussion to queries on single relations.

3 Problem Formulation

In this section, we formulate the central problem being addressed in this paper, namely providing highly-accurate answers to group-by queries in an approximate query answering system. First, we present some relevant background on group-by queries.

[6] Work is also in progress to automatically extract this information from a query workload and adapt the statistics dynamically.

[7] The confidence level is a parameter in Aqua.

3.1 Background

The central fact tables in a data warehouse contain several attributes that are commonly used for grouping the tuples in order to aggregate some measured quantities over each group. We call these the *dimensional* or *grouping* attributes. The attributes used for aggregation are called *measured* or *aggregate* attributes. For example, consider the central table (say, `census`) in a Census database containing the following attributes for each individual (the attribute names are listed in brackets): *social security number* (`ssn`), *state of residence* (`st`), *gender* (`gen`), and *annual income* (`sal`). In this schema, the grouping columns are `st` and `gen`, whereas the aggregate column is `sal`. A typical group-by query on census may request the average income of males and females in each state.

Of course, every query need not involve all the grouping columns in it, e.g., *highest income in each state*. For simplicity, we also consider a query with no groupings as a group-by query returning a single group. It is easily seen that for a relation containing a set G of grouping attributes, there are exactly $2^{|G|}$ possible groupings (the power set U of G) that can occur in a query. In the census relation, G is {`st`, `gen`} and U is $\{\emptyset, (\mathtt{st}), (\mathtt{gen}), (\mathtt{st}, \mathtt{gen})\}$ ($\emptyset$ is the empty set).

Next, we identify the typical requirements of approximate answers to a group-by query and describe natural metrics to quantitatively capture the errors in those answers.

3.2 Requirements on Group-by Answers

For queries returning a single numerical value (e.g., aggregate queries with no group-bys), it is straightforward to define the quality of the answer. It is simply the absolute or relative difference between the exact and approximate answers. However, since group-by queries produce multiple aggregates, one for each group, the metric is not so straightforward. The MAC error presented in [IP99] for quantifying the error in set-valued query answers works by matching the closest pairs in the exact and approximate answers and then suitably aggregating their differences. However, it is inadequate for our purpose because it does not necessarily match corresponding groups in the two answers. Hence, we develop here simple metrics specific to group-by queries.

At a high level, the user has two requirements on the approximate answer to a group-by query. First, the approximate answer should contain *all* the groups that occur in the exact answer, and second, as motivated in the introduction, the estimated answer for every group should be close to the exact answer for that group. We guarantee the first requirement, as long as the query predicates are not too selective, by ensuring that the schemes presented in the paper provide at least minimum-sized samples for every nonempty group in the relation across all grouping attributes.[8] Hence, in the remainder of the paper, we address the second requirement assuming the first to be true. Below, we formally describe simple metrics for capturing this requirement.

Let Q be a group-by query with an aggregate operation on one of the aggregate attributes C. Let $\{g_1, .., g_n\}$ be the set of all groups occurring in the exact answer to the query. Finally, let c_i and c_i' be the exact and approximate aggregate values over C in the group g_i. Then, the *error* ϵ_i *in group* g_i is defined to be the percentage relative error in the estimation of c_i, i.e.,

$$\epsilon_i = \frac{|c_i - c_i'|}{c_i} \times 100. \tag{1}$$

For concreteness, we select a specific formalization, namely relative error, although other similar formulations (e.g., using absolute error) will not change the nature of the problem. We define the error in a group-by query as follows, considering three possible error metrics:

Definition 3.1 *The error ϵ over the entire group-by query returning a set of groups $\{g_1, .., g_n\}$ is defined to be either $\epsilon_\infty = MAX_{i=1}^{n}\epsilon_i$, $\epsilon_{L1} = \frac{1}{n}\sum_{i=1}^{n}\epsilon_i$, or $\epsilon_{L2} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\epsilon_i^2}$.*

Note that this definition applies even to the case of non-group-by aggregate queries, where the result is essentially an aggregate over a single group, in which case the three metrics are the same.

Using this definition as the basis, we can then informally define the primary goal to be one of minimizing one or all of the above errors for a mix of group-by queries.

4 Solutions

In this section we translate the general requirements of an approximate query answering system presented in the previous section to formal criteria on a sampling-based system. Then, we propose solutions for precomputing samples that optimize the criteria for various sets of group-by queries.

We first study individual groups in the answer and then the entire group-by query answer.

4.1 Sampling Requirements for Individual Groups

Here, we discuss the importance of the number of samples on which the aggregate is performed to the accuracy of a sampling-based result. Then, we show that among all possible sampling procedures, uniform sampling maximizes the expected value of this number.

Importance of Sample Size: The approximate answer provided from a sample is a random estimator for the exact answer, and we would like the estimates it produces to have small relative error (Eq. 1) with high probability. In the sampling literature, this quality is typically captured by the *standard error* of an estimator. Consider for example a column C in a relation of size N whose attribute values are $y_1, \ldots, y_N$, and let U be a uniform random sample of the y_i's of size n. Then the sample mean $\bar{y} = \frac{1}{n}\sum_{y_i \in U} y_i$ is an unbiased estimator of the actual mean $\bar{Y} = \frac{1}{N}\sum_{i=1}^{N} y_i$, with a standard error of

$$\frac{S}{\sqrt{n}}\sqrt{1 - \frac{n}{N}}, \tag{2}$$

where

$$S = \sqrt{\frac{\sum_{i=1}^{N}(y_i - \bar{Y})^2}{N-1}}$$

(see, e.g., [Coc77]). In general, the standard error depends on the sample size, the query (aggregate and predicate), and the variance of the expression on which the aggregate is taken. However, query information is usually not known a priori, and even where partial knowledge is available, optimizing for those queries may jeopardize the performance

[8] The only way to ensure this requirement for highly selective queries is to sample nearly the entire relation. Otherwise, none of the sampled tuples may satisfy the predicate. This places a lower bound on the space allocated for samples, as a function of the number of groups and the target selectivity threshold.

for other ad hoc queries. Because of this, short of sampling the entire relation, which is impractical, it is not possible to collect a single sample that works best for all queries. Hence, we first focus on techniques that are used when the aggregate, variance, and the predicate are unknown and later extend the techniques to use this information in Sections 4.7 and 8.

It is clear from the above equation that the standard error is inversely proportional to $\sqrt{n}$ for uniform sampling[9]. This is also true under other common quality measures such as Hoeffding and Chebyshev bounds, which when applied to AVG, COUNT, or SUM queries, are inversely proportional to $\sqrt{qn}$ or $q\sqrt{n}$, where q is selectivity of the predicate. Hence, a natural objective is to maximize the sample size for the group:

Objective: Let Q be an aggregate query with predicate P. In order to maximize the quality of an approximate answer for an aggregate in Q, we seek to maximize the number of sample tuples satisfying P.

Importance of Uniform Sampling: Here, we establish the need to use uniform random sampling for a single group by showing that it maximizes the expected sample size over all query predicates. First, we define some useful terms.

Let $\{t_1, t_2, \ldots, t_N\}$ be the set of N tuples in a relation R. We define a *sampling procedure* to be an assignment to each t_i of a probability p_i, the probability that t_i is selected for the sample. Let $\mathcal{C}_n$ be the class of all such sampling procedures such that $\sum_{i=1}^{N} p_i = n$, i.e., those with expected sample size n. Let $U_n \in \mathcal{C}_n$ be the uniform sampling procedure, i.e., $p_i = n/N$ for all i. A predicate P defines a subset $P(R)$ of R comprised of those tuples satisfying P. For a given predicate P and sampling procedure $C_n \in \mathcal{C}_n$, let $\mathrm{E}[n|P(R)]$ be the expected number of tuples satisfying the predicate in a sample produced by C_n, i.e., $\mathrm{E}[n|P(R)] = \sum_{i:t_i \in P(R)} p_i$. A natural goal, given the above objective, is to maximize the minimum $\mathrm{E}[n|P(R)]$ over all subsets $P(R)$ of a given size.[10] The next lemma shows that the uniform sampling procedure optimizes this goal.

Lemma 4.1 *For each subset size k, $0 < k < N$, the uniform sampling procedure U_n is the unique sampling procedure in $\mathcal{C}_n$ that maximizes the minimum $\mathrm{E}[n|P(R)]$ over all subsets $P(R)$ of size k.*

Proof. For U_n, the minimum $\mathrm{E}[n|P(R)]$ over all subsets $P(R)$ of size k is kn/N (all subsets have this same $\mathrm{E}[n|P(R)]$). For any other sampling procedure in $\mathcal{C}_n$, the reader can readily verify that the subset $P'(R)$ comprised of the k smallest p_i will have $\mathrm{E}[n|P'(R)] < kn/N$. ∎

In summary, we have established that it is important to collect as many uniformly sampled tuples as possible for any single group in query answer. Next, we extend our discussion to the multiple groups occurring in the group-by query answer.

4.2 Sampling Requirements for the Entire Group-by Answer

Recall from Definition 3.1 that the error in an approximate answer to a group-by query is the norm of the errors for the individual groups, for either the L_∞, L_1, or L_2 average norm. Hence, similar to the case of a single group, the quality of an estimator for a group-by query can be measured by the norm of the *standard error* for the individual groups. We seek to allocate a given sample space among the groups so as to minimize this norm.

Consider the L_∞ average norm (the other two norms lead to the same optimal strategy, as discussed in the full paper [AGP99b]). For this norm, and based on our objective, we seek to *maximize the minimum (expected) number of sample tuples satisfying the predicate in any one group*, which we denote by α. We extend our earlier notation and derive an expression for α as follows. Let g be the number of groups. For a relation R, let R_j be the set of tuples in R in group j. A predicate P defines a subset $P(R) = P(R_1) \cup \cdots \cup P(R_g)$. Let $\mathcal{A}_{n,g}$ be the class of all possible allocations of sample sizes to g groups, where the total size allocated is n. For a given predicate P, a sampling allocation $A_{n,g} \in \mathcal{A}_{n,g}$, and a sampling procedure $C_n \in \mathcal{C}_n$, α is given by:

$$\alpha = \min_{j=1,\ldots,g} \{\mathrm{E}[n_j|P(R_j)]\} \tag{3}$$

where $A_{n,g}$ assigns sample size n_j to group j, and the sample within each group j is produced according to C_{n_j}.

For purposes of the analysis that follows, we restrict our attention to predicates that are independent of the groupings, i.e., the predicate's per-group selectivities are the same for all groups.[11] It is clear that our goal is to *maximize* α. Next, we present an optimal sampling strategy for realizing this goal.

Theorem 4.2 *Let T be a set of grouping attributes that partitions a relation R into g non-empty groups, and let X be the available sample space.*[12] *For each predicate of selectivity q, $0 < q < 1$, among all allocations in $\mathcal{A}_{X,g}$ and all sampling procedures in $\mathcal{C}_{n_j}$, the following strategy maximizes the α in Eq. 3 over all subsets $P(R)$ with per-group selectivity q:*

> *S1: Divide the available sample space X equally among the g groups, and take a uniform random sample within each group.*

Proof. It follows from Lemma 4.1 that uniform random sampling within each group maximizes α, for a given allocation strategy. With uniform sampling, each group R_j allocated n_j space has $\mathrm{E}[n_j|P(R_j)] = qn_j$. Hence α is determined by the smallest n_j. Allocating equal space to each group maximizes the smallest n_j, and hence maximizes α. ∎

In the remainder of this section, we consider mapping the strategy $S1$ to various classes of group-by queries, with arbitrary mixes of groupings. The difference between the resulting solutions can be shown by considering an example of grouping by U.S. states. The first solution we discuss would sample from each state in proportion to the state's population, whereas the second would sample an equal number from each state. Considering the two branches of the U.S. Congress, the former is analogous to the House

[9] While we do not analyze other kinds of sampling procedures within a group, it is intuitively clear that sample size will have a positive effect on their accuracy as well.

[10] Note that for all sampling procedures in $\mathcal{C}_n$, the *average* $\mathrm{E}[n|P(R)]$ over all $P(R)$ of a given size k is the same, i.e., $\binom{N-1}{k-1} \cdot n$.

[11] In general, it is not possible to tailor a strategy for a precomputed sample that works best for all predicates, if the per-group selectivities of a single predicate can vary widely. Although the assumption of predicate independence may not always hold in real life, the sample strategy we derive from this analysis works well even when the assumption does not hold.

[12] Throughout this paper, a unit of space can hold a single sampled tuple.

of Representatives while the latter is analogous to the Senate. The other techniques are hybrid extensions and combinations of these two, *a la* the U.S. Congress.

4.3 House

Consider applying the strategy $S1$ to the class of aggregate queries without group-bys. In this case, we have but a single group, so according to $S1$, we take a uniform random sample of size X of the entire relation, as is typically done in traditional sampling procedures. Next, we list two desirable trends of *House* (in general, uniform random samples) which coincide with a user's expectations on the quality of approximate answers.

1. *For the same aggregate operation, the quality of approximate answers increases with the query selectivity.* E.g., the standard error for an estimated average income over the entire nation is typically much smaller than the standard error for one of the states. (An exception would be if the variance among the incomes in a state was markedly smaller than the variance over all the states.)
2. *Answers to queries with the same aggregate and equal selectivities will typically have similar quality guarantees.* Thus assuming an equal number of men and women in the nation, the guarantees for the estimated average incomes for men are typically very similar to the guarantees for the women. (Again, an exception would be if the variances were markedly different.)

4.4 Senate

Consider applying the strategy $S1$ to the class of aggregate queries with the same set T of grouping attributes. For a given relation R, these attributes define a set, $\mathcal{G}$, of nonempty groups. Let m_T be the number of groups in $\mathcal{G}$. By following $S1$, for each nonempty group $g \in \mathcal{G}$, we take a uniform random sample of size X/m_T from the set of tuples in R in group g.[13] For example, if $T = state$ in a US census database, then $\mathcal{G}$ is the set of all states, $m_T = 50$, and we take a uniform random sample of size $X/50$ from each state.

Next, we illustrate a desirable characteristic of the *Senate* samples. *Given a Senate sample for group-by queries involving an attribute set T, we can also provide approximate answers to group-by queries on any subset T' of T, with at least the same quality.* This is because any group on T' contains in it one or more groups on T. Hence it will have at least as many sample points as any group in T, and correspondingly the same or better performance.

Problems with *House* and *Senate*: Note that using the samples from *House* here would result in very few sample points for small groups, and hence in a very small α. On the other hand, *Senate* allocates fewer tuples to the large groups in T than *House*. Hence, whenever queries are uniformly spread over the *entire* data, more of them occur in the large groups, and *House* will perform better than *Senate* for those cases. Next, we present techniques that perform well over larger classes of group-by queries.

4.5 Basic Congress

Here, we apply the strategy $S1$ to the class of aggregate queries containing group-by queries grouping on a single set T of attributes and queries with no group-bys at all. A natural solution is to simply collect both the *House* and the *Senate* samples (analogous to the U.S. Congress). However, this doubles the sample space. Thus, we reduce this factor of 2 by the following strategy.

Let $\mathcal{G}$ be all the non-empty groups in the grouping on T, and let $m_T = |\mathcal{G}|$. Let g be a group in $\mathcal{G}$ and X be the available sample space. Let n_g be the number of tuples in the relation R in group g. Let h_g and s_g be the (expected) sample sizes allocated to g under *House* and *Senate* respectively. Then, under our new approach, we allocate the higher of these two (i.e., $\max(h_g, s_g)$) to g.[14] Of course, this may still result in a total space of X' that is larger than X (one can easily show that $X' \leq \frac{2m_T-1}{m_T}X - m_T + 1 < 2X$). Hence, we uniformly scale down the sample sizes such that the total space still equals X. The final sample size allocated to group g is given by:

$$c_g = X \frac{\max\left(\frac{n_g}{|R|}, \frac{1}{m_T}\right)}{\sum_{j \in \mathcal{G}} \max\left(\frac{n_j}{|R|}, \frac{1}{m_T}\right)}$$

A *Basic Congress* sample is constructed by selecting a uniform random sample of size c_g for each group g in $\mathcal{G}$.

As an example, consider a relation R with two grouping attributes A, B. The different values in these attributes are depicted in the first two columns of Figure 5. Assume that the number of tuples for the groups $(a_1, b_1), (a_1, b_2), (a_1, b_3), (a_2, b_3)$ are 3000, 3000, 1500, and 2500 respectively. The next two columns depict the space allocated by *House* and *Senate* with $T = \{A, B\}$ and $X = 100$. The fifth column depicts the space allocated by *Basic Congress* (before scaling down) by choosing the maximum of the *House* and *Senate* allocations for each group. The next column shows the allocation scaled down to fit the total available space. Note that while *House* allocates less space for the small group and *Senate* allocates less space for the large groups, *Basic Congress* solves both these problems. On the other hand, by considering only the extreme groupings, *Basic Congress* fails to address the sample size requirements of groupings on subsets of T. For example, grouping on A alone would require an optimal allocation of 50 and 50 samples to the two groups a_1 and a_2, whereas *Basic Congress* applied to T allocates 77.3 and 22.7 units of space respectively. Consequently, using *Basic Congress* to answer an aggregate query grouped solely on A would likely lead to a more inaccurate estimate on the group a_2.

We address this problem by our final technique, *Congress*, proposed next.

4.6 Congress

In this approach, we consider the entire set of possible group-by queries over a relation R, i.e., queries grouping the data on any subset (including $\emptyset$) of the grouping attributes, G, in R. Taking a naive approach of applying Strategy $S1$ using space X on each such grouping would result in a space requirement of $2^{|G|}X$. Hence, we perform an optimization similar to *Basic Congress* above, but this time over all possible groupings — not just G and $\emptyset$, as in *Basic Congress*.

[13] Recall that for simplicity we assume throughout this paper that each group is larger than the number of samples drawn from it. Handling scenarios when this is not the case is straightforward.

[14] We also consider an alternative approach, as follows. Let $Y = X/(\sum_{j \in \mathcal{G}} \max(\frac{n_j}{|R|}, \frac{1}{m_T}))$. Take a uniform sample of size Y of the relation R. Let x_g be the number of sampled tuples from a group g. For each group g such that $x_g < Y/m_T$, where m_T is the number of nonempty groups, add to the sample $Y/m_T - x_g$ additional tuples selected uniformly at random from the set of tuples in R in group g. Due to the choice of Y, the expected size of the resulting sample is X. In practice, the difference between the two approaches is negligible.

A	B	*House* $s_{g,\emptyset}$	*Senate* $s_{g,AB}$	*Basic Congress* (before scaling)	*Basic Congress*	$s_{g,A}$	$s_{g,B}$	*Congress* (before scaling)	*Congress*
a_1	b_1	30	25	30	27.3	20 (of 50)	33.3	33.3	23.5
a_1	b_2	30	25	30	27.3	20 (of 50)	33.3	33.3	23.5
a_1	b_3	15	25	25	22.7	10 (of 50)	12.5 (of 33.3)	25	17.7
a_2	b_3	25	25	25	22.7	50	20.8 (of 33.3)	50	35.3

Figure 5: Expected sample sizes for various techniques, for $X = 100$.

Let $\mathcal{G}$ be the set of non-empty groups under the grouping G. The grouping G partitions the relation R according to the cross-product of all the grouping attributes; this is the finest possible partitioning for group-bys on R. Any group h on any other grouping $T \subset G$ is the union of one or more groups g from $\mathcal{G}$. We denote each such g to be a *subgroup* of h. For example, in Figure 5, $G = \{A, B\}$, $\mathcal{G} = \{(a_1, b_1), (a_1, b_2), (a_1, b_3), (a_2, b_3)\}$, and for the grouping $T = \{A\}$, the set of tuples in the group $h = a_1$ is the union of the tuples in the subgroups (a_1, b_1), (a_1, b_2), and (a_1, b_3) of h.

To construct *Congress*, we first consider applying $S1$ on each $T \subseteq G$. Let $\mathcal{T}$ be the set of non-empty groups under the grouping T, and let $m_T = |\mathcal{T}|$, the number of such groups. By $S1$, each of the non-empty groups in T should get a uniform random sample of X/m_T tuples from the group. Thus for each subgroup g in $\mathcal{G}$ of a group h in $\mathcal{T}$, the expected space allocated to g (from considering T) is simply

$$s_{g,T} = \frac{X}{m_T} \cdot \frac{n_g}{n_h}, \tag{4}$$

where n_g and n_h are the number of tuples in g and h respectively. Then, for each group $g \in \mathcal{G}$, we take the maximum over all T of $s_{g,T}$ as the sample size for g, and of course scale it down to limit the space used to X. The final formula is:

$$\text{SampleSize}(g) = X \frac{\max_{T \subseteq G} s_{g,T}}{\sum_{j \in \mathcal{G}} \max_{T \subseteq G} s_{j,T}} \tag{5}$$

For each group g in $\mathcal{G}$, we select a uniform random sample of size SampleSize(g). Thus we take a stratified, biased sample in which each group at the finest partitioning is its own strata.

The space allocation by *Congress* for $G = \{A, B\}$ is depicted in the last two columns of Figure 5 before and after scaling. Each entry in the "before scaling" column is the maximum of the corresponding entries in the $s_{g,\emptyset}$, $s_{g,A}$, $s_{g,B}$, and $s_{g,AB}$ columns. These $s_{g,T}$ contain the optimal allocations according to $S1$ when considering grouping solely on T. By taking the row-wise maximum and then scaling down all values by the same amount

$$f = \frac{X}{\sum_{j \in \mathcal{G}} \max_{T \subseteq G} s_{j,T}}, \tag{6}$$

we ensure that the sample size for *every* group across *all* combinations of group-by columns is within a factor of at most f of its target optimal allocation.[15] Thus *Congress* essentially guarantees that both large and small groups in all groupings will have a reasonable number of samples.

[15] The scale down factor f ranges from 1 (for a uniform distribution across all possible groups at the finest level of grouping G) to almost $2^{-|G|}$ (for a carefully constructed pathological distribution presented in the full paper [AGP99b]).

Key	Grouping Columns			Aggregate Column
K	A	B	C	Q
k_1	a_1	b_1	c_1	q_1
k_2	a_1	b_1	c_2	q_2

Figure 6: Relation Rel with two example tuples

```
select A,B, sum(Q)
from Rel
group by A,B;
```

Figure 7: User Query Q_2

4.7 Adapting to Query Workload

In the full paper, we discuss how to extend the previous strategies to handle preferences between groupings and/or between groups, whenever they can be determined.

5 Rewriting

In Section 2, we demonstrated how Aqua rewrites queries in the presence of uniform random samples. However, that approach does not apply to the biased samples presented in this paper. This section highlights some of the implementation issues that arise when using such samples. We first give some background on generating approximate answers from biased samples. Then, we present different strategies for rewriting queries in the presence of biased samples.

5.1 Approximate Answers from Biased Samples

Recall that query rewriting involves two key steps: a) scaling up the aggregate expressions and b) deriving error bounds on the estimate. The desired formulas for both steps can be derived using standard techniques. We illustrate by focusing on scaling. In Figure 2, the SUM operator was scaled by a factor of 100 since bs_lineitem was a 1% uniform random sample. We refer to this factor as the *ScaleFactor*. However, biased samples are not uniform samples — instead they are a union of different sized uniform random samples of various groups in the relation. Consider Figure 6. It shows a five column table on which the user poses the query Q_2 (Figure 7). Let SampRel be a biased sample of relation Rel, and let the groups $\langle A = a_1, B = b_1, C = c_1\rangle$ and $\langle A = a_1, B = b_1, C = c_2\rangle$ be represented in SampRel by a 1% and 2% sample respectively. Since both groups contribute to the group $\langle A = a_1,\ B = b_1\rangle$ in the answer for Q_2, we have a non-uniform sample from which we must produce an approximate answer. This raises the concern that we may not be able to extract an unbiased estimator for the sum for this group.

However, using standard techniques for estimators based on stratified samples, we can generate an unbiased answer using all the tuples in the biased sample [Coc77]. For each tuple, let its scale factor *ScaleFactor* be the inverse of the sampling rate for its strata. For the SUM operator, we scale

K	A	B	C	Q	SF

(a) SampRel schema

```
select A,B, sum(Q*SF)
from SampRel
group by A,B;
```

(b) Rewritten Query Q_2

Figure 8: *Integrated* Rewriting

K	A	B	C	Q

(a) SampRel schema

A	B	C	SF

(b) AuxRel schema

```
select SR.A, SR.B, sum(Q*SF)
from SampRel SR, AuxRel AR
where SR.A = AR.A and SR.B = AR.B and SR.C = AR.C
group by SR.A, SR.B;
```

(c) Rewritten query Q_2

Figure 9: *Normalized* Rewriting

each value being summed by its *ScaleFactor*, and then sum the result. In query Q_2, for example, we would scale q_1 by 100 and q_2 by 50, and then add up the scaled sum. For the COUNT operator, we sum up the individual *ScaleFactors* of each tuple satisfying the query predicate. For the AVG operator, we compute the scaled SUM divided by the scaled COUNT.

Note that this approach is superior to subsampling all groups down to a common sampling rate in order to apply techniques for uniform sampling. For example, if the sampling rate for a group is j orders of magnitude smaller than the sampling rate for other groups, then the relative error bound for a COUNT operator using Hoeffding bounds can be $j/2$ orders of magnitude worse.

5.2 Rewriting Strategies

We now consider various strategies for rewriting queries to incorporate the scaling discussed above, using the example of the SUM operator. Rewriting strategies for other aggregate operators and error bounds can be derived similarly and are presented in the full paper [AGP99b].

Note that all sample tuples belonging to a group will have the same *ScaleFactor*. Thus, the key step in scaling is to be able to efficiently associate each tuple with its corresponding *ScaleFactor*. There are two approaches to doing this: a) store the *ScaleFactor*(SF) with *each tuple* in SampRel and b) use a separate table AuxRel to store the *ScaleFactors* for the groups. These two approaches give rise to three techniques described below.

The first approach is highlighted in Figure 8. The rewrite technique, called *Integrated*, incurs a space overhead of storing the *ScaleFactor* and a multiplication operation for *every tuple*. However, this approach incurs significant maintenance overhead — insertion or deletion of tuples from SampRel requires updating the *ScaleFactor* of all tuples in the affected groups.

The second approach addresses the maintenance problem by normalizing the SampRel table and is demonstrated in technique *Normalized* shown in Figure 9. It has only marginal maintenance overhead since the *ScaleFactor* information is isolated to AuxRel and thus, updates to SampRel requires updates only to AuxRel. Since the number of groups would very likely be much fewer than the number of tuples, AuxRel would have a lower cardinality than SampRel. However, this approach has an execution time penalty due to the join required between SampRel and AuxRel. Moreover, the join condition can be non-trivial if there are many grouping attributes. The *Key-normalized* technique attempts to minimize this overhead. Since each group is specified explicitly by the attributes values of the grouping columns, they can be replaced by a unique *group identifier* (GID) as shown in Figure 10. Note that this optimization still limits changes to the smaller AuxRel relation during updates and also reduces the space overhead of AuxRel.

K	A	B	C	Q	GID

(a) SampRel schema

GID	SF

(b) AuxRel schema

```
select A,B, sum(Q*SF)
from SampRel, AuxRel
where SampRel.GID = AuxRel.GID
group by A,B;
```

(c) Rewritten Query Q_2

Figure 10: *Key-normalized* Rewriting

K	A	B	C	Q	SF

(a) SampRel schema

```
select A,B, sum(SQ*SF)
from (select A, B, SF, sum(Q) as SQ
        from SampRel
        group by A, B, SF)
group by A,B;
```

(b) Rewritten Query Q_2

Figure 11: *Nested-integrated* Rewriting

In each of the above approaches, the *ScaleFactor* multiplication operation was performed for every tuple. However, since all tuples belonging to a group have the same *ScaleFactor*, one can optimize further to first aggregate over each group and then scale this aggregate appropriately by the *ScaleFactor*. This approach, however, requires a nested group-by query. While applicable to all the three prior techniques, for space limitations we show this optimization in Figure 11 for *Integrated* rewriting and call it *Nested-integrated*.

In Section 7, we compare the query execution speeds of these four approaches.

6 Computation and Maintenance

In the full paper [AGP99b], we present one-pass algorithms for constructing the various biased samples presented in this paper. We also show how to maintain them in the presence of insertions of new tuples into the relation, without accessing the stored relation.

7 Experiments

We conducted an extensive set of experiments to evaluate the various sample allocation techniques and rewriting strategies. The sampling allocation schemes studied were *House*, *Senate*, *Basic Congress*, and *Congress* (Section 4). The rewriting strategies studied were *Integrated*, *Nested-integrated*, *Normalized*, and *Key-normalized* (Section 5). In this section, we present a representative subset of the results generated. They were chosen to show the tradeoffs among these schemes. First, we describe the experimental testbed. Then we perform experiments to measure the accuracy fo the various sample allocation scheme. Finally, we study the performance of the various rewriting strategies.

Attribute	l_id	l_returnflag	l_linestatus	l_shipdate	l_quantity	l_extendedprice
Data Type	int (1, 2, ..)	int	int	date	float	float
Role of Attribute	Primary Key	Grouping			Aggregation	

Figure 12: The **Lineitem** Schema Used in the Experiments

Q_{g2}	Q_{g3}	Q_{g0}
`select l_returnflag, l_linestatus, sum(l_quantity), sum(l_extendedprice) from lineitem group by l_returnflag, l_linestatus;`	`select l_returnflag, l_linestatus, l_shipdate, sum(l_quantity) from lineitem group by l_returnflag, l_linestatus, l_shipdate;`	`select sum(l_quantity) from lineitem where` ($s \leq$ l_id $\leq s + c$);

Table 1: Queries studied

7.1 Testbed

We ran the experiments on Aqua, with Oracle (v7) as the back-end DBMS. Aqua was enhanced to use the proposed allocation schemes to compute its samples and also, the different rewriting strategies.

7.1.1 Database and Queries

In our experiments, we used the database and queries supplied with the TPC-D benchmark. The TPC-D benchmark models a realistic business data warehouse, with sales data from the past six years. It contains a large central fact table called `lineitem` and several much smaller dimension tables [TPC99]. As mentioned in Section 2, it is sufficient to consider queries on a single relation to evaluate the proposed techniques in Aqua. Hence, we restrict our discussion to queries on the `lineitem` table. The schema of this table is given in Figure 7,[16] along with the grouping (dimensional) and aggregation (measured) attributes. In all our experiments, the *Senate* technique computes the samples for the grouping on {l_returnflag, l_linestatus, l_shipdate}.

Next, we extended the TPC-D data to model several relevant aspects of realistic databases. Specifically, consider the groups obtained by grouping the above relation on all the three grouping attributes. In the original TPC-D data, these groups were nearly identical in size. The data in the aggregate attributes was also uniformly distributed. In our experiments, we introduced desired levels of skew into the distributions of the group-sizes and the data in the aggregated columns. This was done using the Zipf distribution, which is known to accurately model several real-life distributions. By changing the z-parameter of the distribution from 0 to 1.5, we are able to generate group-size distributions that are uniform (i.e., all sizes are same) or progressively more skewed. We fixed the skew in the aggregated column at $z = 0.86$, a commonly used z-parameter because it results in a $90-10$ distribution. Finally, we also varied the number of groups in the relation (from 10 to $200K$). For a given number of groups, we generated equal number of distinct (randomly chosen) values in each of the grouping columns. Since the total number of groups is the product of these counts, if the number of groups is n, the number of distinct values in each of these columns becomes $n^{1/3}$.

The different parameters used in our experiments are listed in Table 2. The size of the sample, determined by parameter SP, is given as a percentage of the original relation. In all our experiments, unless otherwise mentioned, the parameter takes its default value listed in the table.

[16] The original `lineitem` table has some other columns which are not relevant to this discussion. We introduced a l_id attribute to the table to use in the experiments.

Parameter	Range of Values	Default
Table Size (T)	$100K - 6M$ tuples	1M
Sample Percentage (SP)	1% – 75% (% T)	7%
Num. Groups (NG)	$10 - 200K$	1000
Group-size Skew (z)	$0 - 1.5$	0.86

Table 2: Experiment Parameters

Queries: We used queries with different number of group-by columns. They are listed in the Table 1 (the suffixes denote the number of group-bys in the queries). The first two queries are derived from Query 3 in the TPC-D query suite. The third query is parametrized to generate queries with desired selectivities on different parts of the data. Queries Q_{g0} and Q_{g3} represent two ends of the spectrum. The former poses the query over the entire relation whereas the latter causes the finest partitioning on three attributes. Q_{g2}, with two grouping columns, is in between the two extremes. The aim of this study is to identify a scheme that can provide consistently good performance for all the three classes and thus, the entire range.

For the current study, we chose parameter s for Query Q_{g0} randomly between 0 and $950K$ and fixed c at $70K$, and generated 20 such queries. Hence, each query selects about $70K$ tuples, i.e., 7% of the table when T is $1M$.

7.2 Accuracy of Sample Allocation Strategies

In this section, we first compare the accuracies of various sample allocation strategies for group-by and non-group-by queries. Then, we study the sensitivity of the various sampling schemes to size of the sample. In each case, we compute the exact as well as approximate answers for queries Q_{g2}, Q_{g3}, and each of the queries in the set Q_{g0}. For Q_{g2} and Q_{g3}, we define the error as the *average* of the percentage errors for all the groups. For the query set Q_{g0}, we define error as the average of the percentage errors for all the queries. In both cases, the error for a single group is computed using Eq. 1 (Section 3). We also measured the maximum errors and observed that the relative performance of all the techniques was identical to the above average error measures.

7.2.1 Performance for Different Query Sets

In this experiment, we fix the sample percentage at 7% and study the accuracy of various allocation strategies for the three classes of queries. Since each query set aggregates over

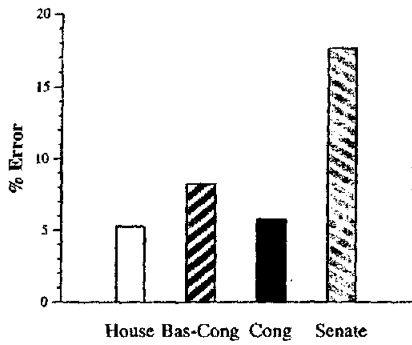

Figure 13: Query Q_{g0} Error

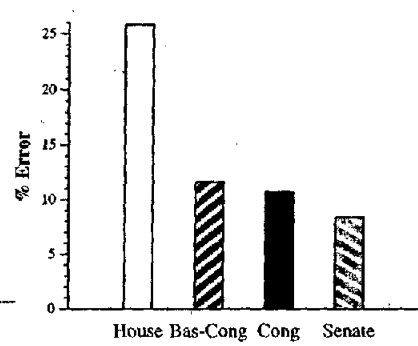

Figure 14: Query Q_{g3} Error

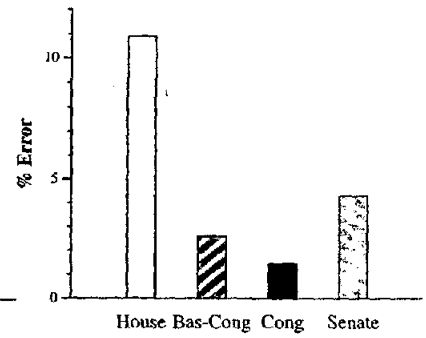

Figure 15: Query Q_{g2} Error

a different set of groups, intuitively, we expect the technique that allocates equal space to those groups to have the least error. Note that, when all the groups are of the same size (i.e., $z = 0$), all the techniques result in the same allocation, which is a uniform sample of the data. Hence, we discuss the results for the case of skewed group sizes (with $z = 1.5$) below.

Queries with No Group-bys (Q_{g0}) (Figure 13): Recall that Q_{g0} consists of queries selecting uniformly over the entire data. Since *Senate* allocates the same space for each group, it ends up allocating less space for the large groups than the other techniques. This results in a higher overall error for *Senate* because a large proportion of the queries land in the large groups. The other techniques perform better because one of their considerations is allocating space uniformly over the entire data. The result is that the space allocation mirrors the queries, and all queries are answered well. The relative performance of these three techniques is determined by the weight they give to this consideration — highest in *House* where it is the sole consideration to the least in *Basic Congress* whose space allocation is skewed towards the small groups. Surprisingly, *Congress*'s errors are low too and it is a good match for *House*.

Queries with Three Group-Bys (Q_{g3}) (Figure 14): Recall that Q_{g3} consists of aggregating over all groups at the finest granularity of grouping. This is precisely the grouping for which the *Senate* sampling was set up giving equal space to each of these groups. Hence, *Senate* has low errors for all the groups resulting in an overall good performance. On the other hand, *House* allocates a large part of the space to the few large groups and incurs high errors for the remaining smaller groups. Once again, *Basic Congress* and *Congress* perform in between these two ranges because they take into account small groups, but to a lesser extent than *Senate*.

Queries with Two Group-Bys (Q_{g2}) (Figure 15): This is the intermediate case of grouping on two attributes. Both *House* and *Senate* perform poorly since they are designed for the two extremes. The absolute magnitude of the error in this case, however, is significantly lower than the last two sets due to the larger size of the groups — both *House* and *Senate* contain enough tuples from each group to produce reasonable estimates. The *Congress* technique easily outperforms them because it is tailored for this case and explicitly considers this grouping in its allocation. Thus, its allocation is close to the ideal for this query set.

Conclusions: It is clear from the above experiments that only *Congress* performs consistently the best or close to best for queries of all types. The other techniques perform well only in a limited part of the spectrum, and thus, are not suitable in practice where a whole range of

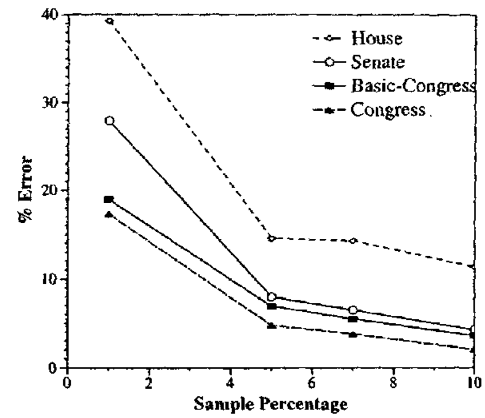

Figure 16: Sample Size vs. Accuracy (Query Q_{g2})

groupings may be of interest to the user during the roll-up and drill-down process. The *Congress* technique performs well because it is not optimized for a particular grouping set but instead takes into consideration all possible groupings (including no-groupings at all) in its space allocation. Thus, even in cases where it is not the best, it is extremely competitive. Consequently, we propose *Congress* as the sampling technique of choice.

7.2.2 Effect of Sample Size

In this experiment we perform a sensitivity analysis test by fixing the group-size skew at 0.86 and measure the errors incurred in answering Query q_{g2} by various allocation schemes for different sample sizes. The results are plotted in Figure 16. As expected, the errors drop as more space is allocated to store the samples. The errors for *House* flatten because it simply allocates more of the available space to the larger groups, which does little to improve the performance for the remaining groups. Overall, the behavior of *Congress* is very encouraging because its errors drop rapidly with increasing sample space. Consequently, it is able to provide high accuracies even for the arbitrary group-by queries.

7.3 Performance of Rewriting Strategies

In these experiments, we measure the actual time taken by each of the four rewriting strategies presented Section 5. We present the time in seconds for running Q_{g2} and writing the result into another relation. The experiments were run on a Sun Sparc-20 with 256MB of memory, and 10GB of disk space running Solaris 2.5. We focus on the effects of sample size and the number of groups because they almost entirely determine the performance of the rewrite strategies.

Technique	Sample Percentage		
	1%	5%	10%
Integrated	1.3	3.8	6.8
Nested-integrated	1.2	3.3	6.0
Normalized	1.7	14.0	27.3
Key-normalized	1.8	14.3	28.4

Table 3: Times Taken for Different Sample Percentages (actual query time = 40sec)

To mitigate the effects of startup and caching, we ran the queries five times and report the average execution times of the last four runs. We present our experiments on sample size; the experiments on the number of groups can be found in the full paper [AGP99b].

Effect of Sample Size on Rewrite Performance: In this experiment, we fix the number of groups at 1000 and vary the sample percentage. Table 3 shows the times taken by various rewrite strategies for different sample percentages. Running the same query on the original table data took 40 seconds on the average. The table makes two points: a) the *Integrated*-based techniques outperform the *Normalized*-based techniques and b) the rise in execution times are dramatic for the *Normalized*-based techniques with increasing sample sizes.

Normalized and *Key-normalized* perform poorly due to the join between the sample table and auxiliary table. Among them, the slightly better performance of *Key-normalized* is due to a shorter join predicate involving just one attribute (l_id), as against two (l_returnflag and l_linestatus) for *Normalized*. Among *Integrated* and *Nested-integrated*, quite surprisingly, the latter performed consistently better in spite of being a nested query. The fewer multiplications with the scalefactor performed by *Nested-integrated* (one per group) pays off over *Integrated* which does one multiplication per tuple. We explore this tradeoff in more detail in the full paper. Overall, solely from the performance viewpoint, these two techniques are still significantly faster than the normalized ones.

Summary of Rewriting Strategies: Our experiments show that *Integrated* and *Nested-integrated* have consistent performance over a wide spectrum of sample sizes and group counts and easily outperform the other two techniques. However, as pointed out in Section 5, they incur higher maintenance costs (which we do not study here). Hence, the choice of a technique depends on the update frequency of the warehouse environment. If the update frequencies are moderate to rare, *Integrated* (or *Nested-integrated*) should be the technique(s) of choice. Only the (rare) high frequency update case warrants for the higher execution times incurred by *Key-normalized*– note that as the warehouse grows larger relative to the sample, the probability of an update reflecting immediately in the sample shrinks significantly, making this an unlikely case in practice.

8 Extensions

In this section, we summarize some extensions to Congressional samples to use different biasing criteria derived from the data and to non-Group-by queries. Details are in the full paper [AGP99b].

Generalization to Multiple Criteria: One of the key features of congressional samples is its extensibility to different space allocation criteria beyond those studied in

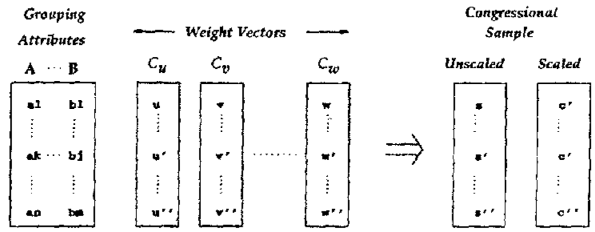

Figure 17: Congressional samples framework

this paper. Consider Figure 17. It shows a typical structure of the table that is used to determine space allocation in a congressional sample similar to that in Figure 5. Note that there are three classes on columns. The ones on the left are the attribute columns which contain the possible groups in some order. The columns in the middle, that we refer to as *weight vectors*, contain for some criteria, the relative ratios of space, or *weights*, to be allocated to each of the groups (e.g., in proportion to the variances of the groups). For example, in Figure 5, *House* and *Senate* strategies contributed a weight vector each. The last two columns aggregate the space allocated by each of the weight vectors to generate the final number of tuples assigned for each group.

Generalization to Other Queries: The Congressional Samples framework can also be extended beyond group-by queries. A group-by query simply partitions the attribute space based on specific attribute values. However, one may also consider other partitions of the space such as ranges of values, where the user has a biased interest in some of the partitions. For example, if a sample of the sales data were used to analyze the impact of a recent sales promotion, the sample would be more effective if the most recent sales data were better represented in the sample as opposed to older data. This can be easily achieved in the above framework by replacing the values in the grouping columns by distinct ranges (in this case on dates) and deriving the weight vectors that weigh the ranges appropriately with respect to each other.

9 Related Work

While statistical techniques based on samples, histograms, etc. have been applied in databases for a while now, they have been primarily used in selectivity estimation during query optimization [SAC+79, Olk93, PIHS96]. Approximate query answering using sampling has started receiving attention recently [Olk93, HHW97, GM98, AGPR99]. The closest work to ours is the *Online Aggregation* scheme proposed by Hellerstein *et al* [HHW97]. In their approach, the original data is scanned in random order at query time to generate increasingly larger random samples of the data, thus incrementally refining the approximate answer generated. Unlike Aqua, that work involves accessing original disk-resident data at query time; but it has the desirable feature of ultimately providing the fully accurate answer. However, both approaches encounter similar problems in answering group-by queries effectively. Their solution is to use the novel the *index striding* technique to control sampling rate among groups and thus ensure fairness among their qualities. Their approach is not suitable for the precomputed or materialized samples considered in this paper.

There have been several recent works using histograms [IP99] or wavelets [VW99] for approximate query answering.

Efficient processing and optimization of aggregate group-by queries has been addressed in [CS94, CS95]. Their techniques are orthogonal to our approach of reducing the data size itself and can be used in Aqua to further speed up group-by query processing.

Biased sampling (e.g., stratified sampling) has been studied in the sampling literature under many contexts [Coc77]. Most related is the work on subpopulation sampling, in which a population is partitioned into subsets (analogous to groups in a group-by query), and on-the-fly sampling is used to estimate the mean or other statistic over each subpopulation, as well as over the entire population. This paper is the first to consider the use of precomputed biased samples for approximate query answering of group-by queries, and extends the previous work by studying combinations of group-by columns, construction and incremental maintenance, query rewriting, optimizing over a range of possible queries, and performance on the TPC-D benchmark data.

10 Conclusions

The growing popularity of OLAP and data warehousing has highlighted the need for approximate query answering systems. These systems offer high performance by answering queries from compact summary statistics, typically uniform random samples, of the data. Needless to say, it is critical in such systems to provide reasonably accurate answers to the commonly posed queries.

In this paper, we showed that precomputed uniform random samples are not sufficient to accurately answer group-by queries, which form the basis of most of the data analysis in decision support systems. We demonstrated that, to be effective for group-by queries, the data should be sampled *non-uniformly*, and proposed several new techniques based on this *biased sampling*. We developed techniques for minimizing errors over queries on a set of possible grouping columns. We introduced *congressional samples*, which are effective for group-by queries with arbitrary group-bys (including none). Additionally, we proposed efficient techniques for constructing congressional samples in one pass over the relation, and for incrementally maintaining them in the presence of database insertions, without accessing the stored relation. We also presented efficient strategies for using the biased samples. The new sampling strategies were validated experimentally both in their ability to produce accurate estimates to group-by queries and in their execution efficiency.

All of the techniques presented in this paper have been incorporated into an approximate query answering system, called Aqua, that we have developed. By providing the ability to answer the important class of group-by queries, our new techniques have significantly enhanced the overall accuracy and usability of Aqua as a viable decision support system. Of course, the techniques themselves are applicable beyond Aqua, and even beyond group-by queries, and can be used wherever the studied limitations of uniform random samples become critical.

Acknowledgements

Sridhar Ramaswamy was one of the designers and implementors of the Aqua prototype. We also thank him for discussions related to this work.

References

[AGP99a] S. Acharya, P. B. Gibbons, and V. Poosala. Aqua: A fast decision support system using approximate query answers. In *Proc. 25th International Conf. on Very Large Databases*, pages 754–757, September 1999. Demo paper.

[AGP99b] S. Acharya, P. B. Gibbons, and V. Poosala. Congressional samples for approximate answering of group-by queries. Technical report, Bell Laboratories, Murray Hill, New Jersey, November 1999.

[AGPR99] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. Join synopses for approximate query answering. In *Proc. ACM SIGMOD International Conf. on Management of Data*, pages 275–286, June 1999.

[CD97] S. Chaudhuri and U. Dayal. An overview of data warehousing and OLAP technology. *SIGMOD Record*, 26(1):65–74, 1997.

[CMN99] S. Chaudhuri, R. Motwani, and V. Narasayya. On random sampling over joins. In *Proc. ACM SIGMOD International Conf. on Management of Data*, pages 263–274, June 1999.

[Coc77] W. G. Cochran. *Sampling Techniques*. John Wiley & Sons, New York, third edition, 1977.

[CS94] S. Chaudhuri and K. Shim. Including group-by in query optimization. In *Proc. 20th International Conf. on Very Large Data Bases*, pages 354–366, September 1994.

[CS95] S. Chaudhuri and K. Shim. An overview of cost-based optimization of queries with aggregates. *IEEE Data Engineering Bulletin*, 18(3):3–9, 1995.

[GM98] P. B. Gibbons and Y. Matias. New sampling-based summary statistics for improving approximate query answers. In *Proc. ACM SIGMOD International Conf. on Management of Data*, pages 331–342, June 1998.

[HH99] P. Haas and J. Hellerstein. Ripple joins for online aggregation. In *Proc. ACM SIGMOD International Conf. on Management of Data*, pages 287–298, June 1999.

[HHW97] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. In *Proc. ACM SIGMOD International Conf. on Management of Data*, pages 171–182, May 1997.

[IP99] Y. Ioannidis and V. Poosala. Histogram-based techniques for approximating set-valued query-answers. In *Proc. 25th International Conf. on Very Large Databases*, pages 174–185, September 1999.

[Kim96] R. Kimball. *The Data Warehouse Tookit*. John Wiley and Sons Inc., 1996.

[Olk93] F. Olken. *Random Sampling from Databases*. PhD thesis, Computer Science, U.C. Berkeley, April 1993.

[PIHS96] V. Poosala, Y. E. Ioannidis, P. J. Haas, and E. J. Shekita. Improved histograms for selectivity estimation of range predicates. In *Proc. ACM SIGMOD International Conf. on Management of Data*, pages 294–305, June 1996.

[SAC+79] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. T. Price. Access path selection in a relational database management system. In *Proc. ACM SIGMOD International Conf. on Management of Data*, pages 23–34, June 1979.

[Sch97] D. Schneider. The ins & outs (and everything in between) of data warehousing. Tutorial in the *23rd International Conf. on Very Large Data Bases*, August 1997.

[TPC99] Transaction processing performance council (TPC). *TPC-D Benchmark Version 2.0*, February 1999. URL: `www.tpc.org`.

[VW99] J. S. Vitter and M. Wang. Approximate computation of multidimensional aggregates of sparse data using wavelets. In *Proc. ACM SIGMOD International Conf. on Management of Data*, pages 193–204, June 1999.