Answering Aggregation Queries on Hierarchical Web Sites Using Adaptive Sampling[*]

(Technical Report, UCI ICS, August, 2005)

Foto N. Afrati
Computer Science Division
NTUA, Athens, Greece
afrati@softlab.ece.ntua.gr

Paraskevas V. Lekeas
Computer Science Division
NTUA, Athens, Greece
plekeas@mail.ntua.gr

Chen Li
Dept. of Computer Science
UC Irvine, CA 92697, USA
chenli@ics.uci.edu

August 16, 2005

Abstract

Many Web sites publish their data in a hierarchical structure. For instance, Amazon.com organizes its pages about books as a hierarchy, in which each leaf node corresponds to a collection of pages of books in the same class (e.g., books on `Data Mining`). This class can be browsed easily by users following a path from the root to the corresponding leaf node, such as "`Computers & Internet` — `Databases` — `Storage` — `Data Mining`." Business applications often require to ask aggregation queries on such data, such as "finding the average price of books on `Data Mining`." On the other hand, it is computationally expensive to compute the *exact* answer to such a query due to the large amount of data, its dynamicity, and limited Web-access resources. In this paper we study how to answer such aggregation queries approximately with quality guarantees using sampling. We study how to use adaptive-sampling techniques that allocate the resources adaptively based on partial samples retrieved from different nodes in the hierarchy. We study how to estimate the quality of the answer using the sample. Our experimental study using real and synthetic datasets validates the proposed techniques.

1 Introduction

Many Web sites such as e-commerce sites and portals publish their data in HTML pages organized as hierarchies. For instance, Amazon.com has millions of pages about its books. It classifies these books as different hierarchical classes. Figure 1 shows part of the hierarchy. Each internal node in the hierarchy represents a class of books, such as `Accessories` and `History`. This node corresponds to a Web page that displays the categories (Web links) of its child classes. A leaf node contains a collection of books belonging to the corresponding category, such as `Warehousing` and `Encryption`.

[*]Foto N. Afrati and Paraskevas V. Lekeas were supported by the project PYTHAGORAS I, co-funded by the European Social Fund (75%) and National Resources (25%).

These books are published as different pages. The main reason of having such a hierarchy is to allow easy browsing by Web users, who can browse the pages starting from the root to locate the books in a class. For instance, a user can easily find pages of `Data Mining` books by starting from the root page and following the path "`Computers & Internet` — `Databases` — `Storage` — `Data Mining`."

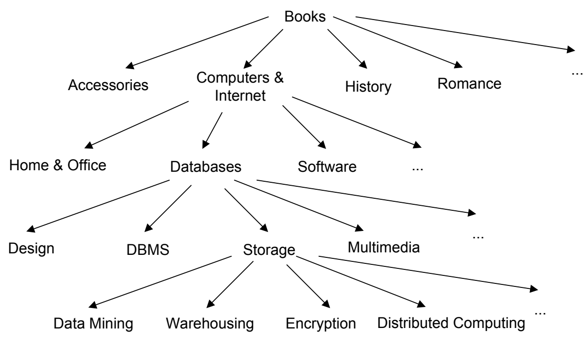

Figure 1: Book hierarchy at Amazon.com.

Many applications built on top of these Web sites need to ask queries on the data. Consider, for example, an online book store that competes with Amazon.com. In order to make marketing strategies about its own books, the company wants to collect the information about the prices of different classes of books at Amazon.com. The following are a few example queries the company would issue about Amazon.com books.

- Query Q_1: Find the average price of books on `Warehousing`. These books correspond to a class of a leaf node.
- Query Q_2: Find the average price of books on `Databases`. These books correspond to a class of an internal node in the hierarchy.

One way to answer such a query is to access all the pages of the books that are relevant to the query. For instance, for query Q_1, we access all the pages of the books on `Warehousing`, and use the prices of these books to compute their average price. For query Q_2, we need to access the pages of `Databases` books. Although this approach can return the exact answer to a query, when there are many books relevant to the query, this method requires to access a large number of pages, causing a significant workload on the Web site. The problem becomes even worse when we need to ask such queries frequently, e.g., when the data at the Web site is very dynamic. If we limit the number of requests sent to the server for a given period, then we need a long period to access all the relevant pages. However, the book information at the server may have changed during the long period, and the retrieved data is already out of date.

In this paper we study how to answer aggregation queries on Web hierarchies by sampling. Given a query, we sample some of the pages of relevant objects to compute an approximate answer to the query. We focus on the following three main challenges related to how to allocate Web-access resources to the pages of different classes, illustrated by the running example on books. First, the books from different classes could have different price distributions. For instance, the number of books and average price of the `Warehousing` class could be very different from those of the `Encryption` and `Distributed Computing` classes. As of October 2004, the numbers of books in these three categories were 122, 132, and 202, respectively. Their average prices were \$336, \$78, and \$91, respectively. Thus, a naive, random-sampling approach is not an ideal method. Second, the distributions of the objects in different categories are not known a-priori. Third, we need to estimate the quality of the answers computed using sampling.

The approach we adopt to deal with these challenges is to use adaptive sampling, in which we decide progressively which nodes to oversample and which nodes to undersample in order to obtain a solution of good quality without wasting resources. The reason that this approach is expected to have good results can intuitively be seen in the following two situations. First, consider the case where we need to answer a single query that asks for the average price in an internal node. Some of its relevant leaf nodes may not contribute significantly to the solution. They might, e.g., contain only a few books of moderate prices compared to the other relevant leaf nodes. Second, when we need to answer a set of queries, some leaf nodes can contribute a lot to the quality of the answers of the queries, while others are relevant only to a few queries. For instance, a leaf node might contain many expensive books on `Landscape Architecture` distributed over a large price range, whereas nodes about `Computer Science` books contain only moderately priced books. Thus if we have a set of queries that ask for average prices of books in `Landscape Architecture` and only a few single queries on `Computer Science`, then in order to maintain good quality, we choose to oversample the `Landscape Architecture` books and undersample the `Computer Science` books.

In this study we make the following contributions.

- We formulate a new problem about answering aggregation queries on hierarchical web sites. We propose adaptive-sampling algorithms for answering a single AVG query, when the node in the query is a leaf node or an internal node.
- We propose adaptive-sampling algorithms for answering a single MAX/MIN query.
- We propose an algorithm for answering a set of aggregation queries.
- We conduct experiments to evaluate these algorithms on both a real data set and a synthetic data set.

The rest of the paper is organized as follows: In Section 2 we give the problem formulation. Section 3 describes our algorithms for answering single AVG queries for leaf nodes and internal nodes. Section 4 presents our algorithms for answering single MAX/MIN queries. Section 5 deals with how to answer a group of aggregation queries. Section 6 reports our experimental results. Section 7 concludes the work.

1.1 Related Work

Sampling techniques have been widely used in different contexts in databases. See [11] for a list of related papers. For instance, [8, 10, 9] addressed the problem of estimating the result size for a query using random sampling without going through the entire result. [4] used sampling to estimate join selectivity. Chaudhuri et. al [1] studied how to overcome limitations of sampling for aggregation queries for distributions of skewed aggregated attributes and queries with a low selectivity. [6] studied how to use sampling techniques to answer aggregation queries online by reporting the progress of answering the aggregation query. Recently there have been a lot of studies on using sampling techniques to answer queries on data streams. See [3] for a good survey.

Compared to these related works, our problem is unique in the sense that we consider Web data published as a hierarchy, and we do not know the distributions of different leaf nodes a priori. We need adaptive sampling techniques to allocate resources effectively and do the corresponding analysis about the quality of the computed results based on the samples.

2 Formulation

In this section we formulate the problem studied in this paper.

Hierarchical Web-Site Structure and Access Model: Consider a Web site that publishes its information about *objects* such as books, electronics, and computers. The information about an object is published in a single page, and one Web access is needed to retrieve this page. The site organizes these pages as a hierarchical tree [5, 2], where each leaf node corresponds to a class of objects. For example, in Figure 1, the leaf node `Data Mining` has a list of links to the pages of data mining books. (This list could be published in multiple pages for browsing purposes.) The *size* of a leaf node is the number of all its objects. An internal node in the hierarchy corresponds to a list of links pointing to its children, which are internal nodes or leaf nodes. The *size* of an internal node is the sum of the sizes of all its descendant leaf nodes. We assume the number of objects in each leaf node is given. Thus the population of an internal can be easily computed.

Queries: We consider aggregation queries using functions AVG, MAX, and MIN. An aggregation query on a node concerns the objects represented by this node and the aggregation is applied on these objects. We call this node the *query node* for the given query. A query node can be a leaf node or an internal node. For example, in Figure 1, if we ask for the average price of all the `Encryption` books, then the query node is a leaf node. If we ask for the maximum price of the `Databases` books, then the query node is an internal node, and the actual data is stored in its descendant leaf nodes such as `DBMS`, `Data Mining`, `Encryption`, etc. Formally, an aggregation query is represented as $\gamma(v)$, in which γ is an aggregation function (AVG, MAX, or MIN) and v is a node in the hierarchy. For instance, the query $MAX(\texttt{Databases})$ asks for the maximum price of all the books in the `Databases` class. Let $ans(\gamma(v))$ denote the real answer to this query.

Approximate Answers with Quality Guarantees: Given a query $\gamma(v)$, we want to compute an approximate answer to the query using samples from the objects in the classes relevant to the query, i.e., those objects in the leaf nodes of the node v. The query also comes with a confidence δ, which is a real between 0 and 1. Our goal is to find an estimation e of the real answer to the query, such that with probability at least $1 - \delta$ the "distance" between the estimated answer and the real answer is as small as possible. The "distance" measurement depends on the aggregation function γ. For instance, in the case of AVG, the distance is $|e - ans(\gamma(v))|$. For the case of MAX and MIN, we use quantiles to measure such a distance (Section 4).

Our methods to approximately answer the query rely on sampling. We view the data in the class of each leaf node as a random variable that takes values uniformly at random from the objects in this class. We assume that Web-access resources are sparse and expensive. We focus on how to make the best use of the available resources to improve the quality of the approximate answer. Specifically, one problem we are investigating is: Given a number of resources and a confidence, what is the best way to sample the objects to answer the query so that we obtain a good approximation of the real answer with the given confidence? Our solutions can also be used to solve other related problems, such as deciding how many resources are needed when the user also specifies a threshold on the distance between the estimate answer and the real answer.

As we do not know the distributions of the objects in different leaf nodes a priori, we use *adaptive sampling*. The main idea is to allocate resources to these leaves proportionally to a measure that represents the importance of the particular leaf in finding a good answer to the query. The measure also represents a parameter of the distribution of the data in the particular leaf. For instance, if the standard deviation of the objects in a particular leaf is small, then we do not need to allocate a large number of resources to this node, because we can obtain a good estimation with a few resources. On the other hand, if the standard deviation is large, we need to allocate more resources to this leaf node in order to achieve a good approximation.

We propose a family of adaptive sampling algorithms for different aggregation functions. Each algorithm allocates the available resources iteratively. In each stage, it assigns resources to those leaves that introduce more error in the estimated answer. The algorithm adaptively decides how many resources need to be allocated to a leaf node in the next iteration, until we use all the resources, or, alternatively, when the user is satisfied with the quality of the approximate answer.

3 Answering AVG Queries

In this section we consider how to answer an AVG aggregation query, denoted as $AVG(v)$, which asks for the average value of the objects belonging to the class of a node v. The query also specifies a confidence δ, and we want to get an average interval with a confidence at least $1 - \delta$.

3.1 AVG Query on Leaf Node

First we consider the case where the node v is a leaf node. We allocate all available resources to the leaf node, and randomly draw samples from it. Then we estimate the answer of the query within a confidence interval. There are two approaches that give us certain quality guarantees. The first one, called the *Hoeffding approach*, is a conservative approach and guarantees that the estimated value is within the confidence interval with probability at least $1 - \delta$. The second one, called the *CLT approach*, is associated with the Central Limit Theorem, and it assumes that the sample is small compared to the size of the population, but large enough so that approximations based on CLT are accurate.

3.1.1 The Hoeffding Approach

Theorem 3.1 *(Hoeffding's Bound [7]) Let $x_1, x_2, \ldots, x_n$ be independent instances of a bounded random variable X, with values in $[a, b]$. Denote their average by $\bar{x}_n = \frac{1}{n}\sum_{i=1}^{n} x_i$, and let μ be the expectation of X. For any $\epsilon > 0$, we have:*

$$Pr\{|\bar{x}_n - \mu| \geq \epsilon\} \leq 2e^{-\frac{2n\epsilon^2}{(b-a)^2}}.$$

□

Based on this theorem, given a set of n random samples $x_1, \ldots, x_n$ of the objects in the leaf node, we can use their average to estimate the real average. To have a confidence $(1 - \delta)$ that this estimated average $\bar{x}_n$ is within ϵ of the real average μ, we get from the above theorem:

$$2e^{-\frac{2n\epsilon^2}{(b-a)^2}} = \delta.$$

Therefore by solving on ϵ we get:

$$\epsilon = \sqrt{\frac{(b-a)^2 log(\frac{2}{\delta})}{2n}}. \quad (1)$$

We may use the same theorem to estimate the number of samples needed to be drawn if we are given a specific confidence interval ϵ:

$$n = \frac{(b-a)^2 log(\frac{2}{\delta})}{2\epsilon^2}. \quad (2)$$

We assume the interval $[a, b]$ is well approximated by considering the maximum value and the minimum value of the set of samples. (A similar approach is taken in the context of approximate estimation in data streams [3].) Given a confidence δ and a positive integer n, Equation 1 tells us how close the average $\bar{x}_n$ of the sampled object is to the real average μ if we draw n random samples from the population. Similarly, given a confidence δ and a confidence interval ϵ, Equation 2 tells us how many random samples are needed.

3.1.2 The CLT approach

Theorem 3.2 *(Central Limit Theorem) Let $\bar{X}$ denote the mean of the observations in a set of random samples of size n from a population having a mean μ and standard deviation σ. Denote the mean of the $\bar{X}$ distribution by $\mu_{\bar{X}}$, and the standard deviation of the $\bar{X}$ distribution by $\sigma_{\bar{X}}$. When n is sufficiently large, the sampling distribution of $\bar{X}$ is well approximated by a normal distribution (even when the population distribution is not itself normal).* □

Based on the Central Limit Theorem (CLT), given n random samples $x_1, x_2, \ldots, x_n$ from the objects in a leaf node, for the estimation of $\bar{X}$ to be within $\pm\epsilon$ of the real average with a confidence δ, we can use the fact that $\bar{X}$ is well approximated by a normal distribution. Using tail inequalities for the normal distribution, we get the error parameter

$$\epsilon = \left(\frac{z_\delta^2 T_{n,2}}{n}\right)^{\frac{1}{2}} \geq 0,$$

where z_δ is the $\frac{(\delta+1)}{2}$ quantile of the sample distribution, and

$$T_{n,2} = \frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n-1},$$

where $\bar{x} = \sum_{i=1}^{n} x_i$.

3.2 AVG Query on Internal Node

Now consider an AVG query for an internal node v. Let v have m descendant leaf nodes $c_1, c_2, \ldots, c_m$ with populations $n_1, n_2, \ldots, n_m$, respectively. (Notice that v might not be the immediate parent node of each such leaf node.) Let $n = n_1 + n_2 + \ldots + n_m$ be the population of the node v. A naive approach to estimating the average on node v is to distribute the available resources equally to all leaves $c_1, c_2, \ldots, c_m$. This approach could waste a lot of resources. If the objects of one leaf have a very small standard deviation, then an accurate estimation of its average can be done by drawing a small set of random samples (e.g., see Equation 2). The spared resources can be used on another leaf with a larger standard deviation, and needs more samples for a tolerable estimation.

We propose an adaptive sampling algorithm (Figure 2) for answering such a query approximately. Its main idea is to assign more resources to the leaf nodes with larger standard deviations. As we cannot know the standard deviation of each leaf in advance, the algorithm goes through a number of iterations. In each iteration, it assigns resources to each node proportionally to its currently estimated standard deviation. With the proper tuning of the number of iterations and the resources assigned in each iteration, the algorithm can compute an approximate answer with a high quality.

We give the details of the algorithm. The algorithm uses several iterations so that it can adapt itself to the current estimation of the standard deviations on the leaves. In each iteration, the estimation of a leaf node c_i may not be accurate. As a consequence, the algorithm assigns too

```
Algorithm: AVG_INTERNAL
Input:
        $AVG(v)$: an AVG query on an internal node $v$;
        $\delta$: confidence;
        $m$: number of leaves in the query node;
        $N$: total number of available resources;
        $d$: number of resources allocated in each iteration.
Output: estimated average with a confidence interval
Method:
    Allocate $d/m$ resources to each leaf of $v$;
    WHILE (there are remaining resources) {
        For (each leaf $c_k$) do {
            Compute the standard deviation $\sigma_k$ of leaf $c_k$.
            Allocate to leaf $c_k$ resources for the next iteration
                proportionally to its standard deviation.
        }
    }
    Using the samples to compute the average, standard
        deviation, and confidence interval of the query.
```

Figure 2: Algorithm for answering an AVG query on an internal node.

many or too few resources on the node c_i. Then in subsequent iterations, the algorithm will adjust the estimations closer to the real values. For example, if it assigned too many resources to leaf c_i, then after a better estimation of the standard deviation of c_i, the algorithm will save resources in subsequent iterations. We set a number d to be the number of resources allocated in each iteration. The algorithm has two phases.

Initial resource allocation: We allocate an equal number of resources to the leaf nodes initially. Thus for m leaves $c_1, c_2, \ldots, c_m$, we assign $\frac{d}{m}$ resources in each leaf.

Resource allocation in the i-th Iteration : It has two stages:

- For each leaf c_k, we compute the average $\bar{\mu}_k$ and the standard deviation $\bar{\sigma}_k$ based on its samples from all previous iterations.
- For each leaf node c_k, it assigns the following number of resources: $\frac{d \cdot \bar{\sigma}_k}{\sum_{i=1}^{m} \bar{\sigma}_k}$, which is proportional to its estimated standard deviation.

The stopping condition of the algorithm is when it runs out of resources. In this case, we have N/d iterations, where N is the total number of available resources. Alternatively, we can stop the iterations when the user is satisfied with the confidence interval of the estimated answer to the query, which is computed as follows.

Estimation of the query in the internal node: We compute the mean $\bar{\mu}_v$ and the standard deviation $\bar{\sigma}_v$ using all the samples. Finally, based on these two values, we use one of the approaches described in Section 3.1 (Hoeffding or CLT) to compute the confidence interval.

4 MAX/MIN Queries

Now we study how to answer MAX/MIN aggregation queries approximately by random sampling. We focus on answering a MAX query, and the results are applicable to the case of answering a MIN query. A MAX query $MAX(v)$ asks for the maximum value of the objects of the class v. The query also defines a confidence δ.

4.1 MAX Query on Leaf Node

Consider the case where the query node in the MAX query is a leaf node. Given n resources, we allocate all of them to the leaf node to draw a set of random samples. We compute the maximum value M of these samples as an approximate answer to the query. In order to measure the quality of this answer, we need the notion of *quantile*. Given a random variable X, its ϕ *quantile* is a value t of the variable such that the probability that X is greater than t is $1 - \phi$. We can use the quantile of the maximum M of the samples to measure the quality of this approximate answer. Ideally we want the quantile to be as close to 1 as possible. To estimate the quantile, we consider the Chebyshev inequality.

Theorem 4.1 *(Chebyshev Inequality) If a random variable X has a finite mean μ and a finite variance σ^2, then for any $\epsilon \geq 0$:*

$$Pr[|X - \mu| \geq \epsilon] \leq \frac{\sigma^2}{\epsilon^2}$$

□

The inequality can be used to give us a bound on the probability that $|X - \mu| \geq M - \mu$ for the maximum value M in the samples. Thus using in the inequality $\epsilon = M - \mu$, we get:

$$Pr[|X - \mu| \geq M - \mu] \leq \frac{\sigma^2}{(M - \mu)^2}$$

$$Pr[X \geq M] \leq \frac{\sigma^2}{(M - \mu)^2} \tag{3}$$

Suppose we knew the average μ and standard deviation σ of the random variable. Equation 3 says that the probability a value of an object relevant to the query is no less than M is at most $1 - \frac{\sigma^2}{(M-\mu)^2}$. In other words, M is at least the $1 - \frac{\sigma^2}{(M-\mu)^2}$ quantile of the values of these objects. We call this quantile the *confidence quantile* for the value M.

Equation 3 assumes that the average and the standard deviation of the random variable are known. We can use the samples to estimate these parameters. We estimate the average using the algorithms in Section 3. As explained earlier, given a confidence, these algorithms return an average value with a confidence interval. Using the set of samples, we can estimate the average and standard deviation of X within a distance ϵ_{avg} and ϵ_{dev} with probability $1 - \delta$, respectively.[1] We use these estimations to compute the confidence quantile for the maximum value M. Since we use these estimations, we have finally that M is greater than the ϕ quantile of X with a probability $1 - \delta$, where ϕ is:

$$\phi = 1 - \frac{\sigma^2 + \epsilon_{dev}^2}{(M - \mu - \epsilon_{avg})^2} \tag{4}$$

Therefore, with a confidence δ, the value M is not less than the ϕ quantile of X.

[1] For the estimation of the average, see the previous section. For the estimation of the standard deviation, a similar argument applies.

4.2 MAX Query on Internal Node

If the node in the MAX query is an internal node, we answer the query similarly to the case we answer an AVG query for an internal node. The difference is that, for each leaf node, its decisive quantity is not its standard deviation but $1 - \phi$, where ϕ is the quantile of the currently estimated maximum value for this leaf. That is, suppose the maximum value of all the samples drawn so far is M. For each leaf node relevant to the query, we do the following. We compute the average and the standard deviation of the samples drawn from this leaf node, and compute the confidence quantile of M with respect to these samples. That is, in Equation 3, we use the average and the standard deviation of the samples from this leaf. At the end of the algorithm, we output the confidence quantile of all the samples. The algorithm is given in Figure 3.

```
Algorithm: MAX Query for an Internal Node
Input:
        $MAX(v)$: a MAX query on an internal node $v$;
        $\delta$: confidence;
        $m$: number of leaves in the query node;
        $N$: total number of available resources;
        $d$: number of resources allocated in each iteration.
Output: estimated max with a confidence quantile
Method:
    Allocate each leaf $d/m$ resources;
    WHILE (there are remaining resources) {
        Compute the max of all samples;
        For (each leaf $c_k$) do {
            Compute the standard deviation $\sigma_k$ and
                confidence quantile $\phi_k$ of $M$ for leaf $c_k$;
            Allocate to leaf $c_k$ resources for the next iteration
                proportionally to its $1 - \phi_k$.
        }
    }
    Using the samples to compute the average, standard
        deviation, and confidence quantile of the query.
```

Figure 3: Answering MAX query on internal node.

5 Answering Multiple Aggregation Queries

In this section we study how to answer a set of aggregation queries given limited resources. As an example, consider the hierarchy shown in Figure 4, on which we are given four AVG queries Q_1, Q_2, Q_3, and Q_4. We are interested in answering these queries as a group when given a limited number of resources to access the objects of the leaf nodes. We develop an algorithm, called *ϵ-shake*, for answering such queries using adaptive sampling. For simplicity, we mainly focus on the case where all the queries are AVG queries. Our results can be extended to the case where the queries have different aggregation functions.

One main challenge is the interrelationships among the queries. For example, node g contributes to the answers to three queries, and node a only to one query. Consider the case where the distribution on node g has a large variance, and the distribution on node a has a small variance. If their populations are the same, then we want to allocate more resources to node g in order to get

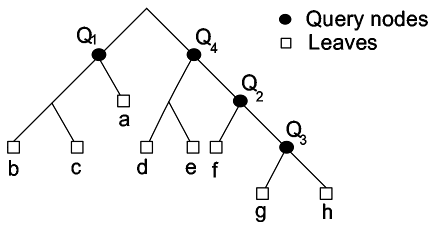

Figure 4: Multiple aggregation queries.

a solution of better quality. However, in the case a has a large variance, it becomes unclear what sampling strategy we should follow.

Similarly to the algorithms presented in previous sections, the ϵ-shake algorithm allocates resources to the leaf nodes iteratively. In each iteration, it uses a qualitative criterion for each leaf node to measure its contribution to the answers to the queries. We produce small vibrations to the leaves, i.e., we "shake the leaves." We check how these vibrations affect the estimation to the query answers. In particular, suppose the quality of the estimation on a certain leaf improves by a small value ϵ, and the estimations by all other leaves remain the same, we compute the new estimation of the answers to the queries. Based on the results, we decide how many resources to allocate to this leaf in the next iteration. The leaves that cause more changes to the current estimation are considered to be more important, and deserve more resources in the next iteration. Notice that since we have a set of queries rather than one, the quality of the solution depends on the quality of the approximate answer to each query. To measure this quality, we need to establish a compound cost that combines the quality measures of all the queries.

Formally, let Q be an AVG query on a node with n leaves, $c_1, \ldots, c_n$. Let $a_1, a_2, \ldots, a_n$ denote the averages of the populations in these leaves. Thus we have

$$Q = f(a_1, a_2, \ldots, a_n) = \frac{\sum_i n_i a_i}{n},$$

where n_i is the population of leaf c_i. The error δQ for estimating Q is given in terms of the error in the estimation of each leaf by the error propagation formula

$$\delta Q = \sqrt{(\frac{\theta f}{\theta a_1}\sigma_{a_1})^2 + (\frac{\theta f}{\theta a_2}\sigma_{a_2})^2 + \ldots + (\frac{\theta f}{\theta a_n}\sigma_{a_n})^2}.$$

Suppose we have estimated Q within δQ and we want to see how a small change, ϵ, in the estimation of one leaf (say, leaf c_1) can affect the previous estimation. For the new estimation error, we have

$$\delta_\epsilon Q_{a_1} = \sqrt{(\frac{\theta f}{\theta a_1}(\sigma_{a_1} - \epsilon))^2 + (\frac{\theta f}{\theta a_2}\sigma_{a_2})^2 + \ldots + (\frac{\theta f}{\theta a_n}\sigma_{a_n})^2}.$$

A measure of this change is a *weight*

$$w_{c_1} = \frac{\delta Q_{a_1} - \delta_\epsilon Q_{a_1}}{\delta Q_{a_1}} = 1 - \frac{\delta_\epsilon Q_{a_1}}{\delta Q_{a_1}},$$

which characterizes leaf c_1. For a fixed ϵ, we can have n weights $w_{c_1}, w_{c_2}, \ldots, w_{c_n}$ for the leaf nodes, leading us to an ordering of these leaf nodes according to their significance of contribution in the query Q. Observe that under this setting, if a leaf node u is irrelevant to the query Q, then it has a zero contribution, since $\frac{\theta f}{\theta u} = 0$ and $\delta Q - \delta_\epsilon Q_u = 0$.

When we have multiple queries, the above idea carries over by taking the combined error measure for each leaf, which is the sum of its weights for each query. Finally we assign resources proportionally to the combined error measure.

6 Experiments

We have conducted experiments to evaluate the proposed algorithms for both a real data set and a synthetic data set. We collected the real data about book prices from Amazon.com from March 2004 to July 2004. We implemented a crawler to download all the "Buy New" price tags of different book categories. For instance, one such category is books of `Database Design`, which included about 800 books. For the synthetic data set, we used Mathematica to produce leaf nodes with the same population and standard deviation as the leaf nodes in the real data set, but with a normal distribution.

6.1 AVG on Leaf Node

We first conducted experiments for AVG queries on leaf nodes of the Amazon book hierarchy, using both the Hoeffding approach and the CLT approach. We used three leaf nodes of different sizes, one node with 369 books, one node with 2157 books, and one node with 11973 books. We set the confidence of an AVG query to be 95%, i.e., $\delta = 0.05$. Each sampling algorithm was repeated 10 times to get a good evaluation of the behavior of the algorithm. Figure 5 shows how the Hoeffding approach behaved for the three leaf nodes, when we sampled about 10% of their population. The x-axis represents the different runs of the algorithm. For each run, the figure shows the estimated average and the corresponding confidence interval (shown as a vertical segment). For the node with a small population (369), its real average is 64. We drew a set of 37 random samples in each run. For different runs, their estimations had a relatively big confidence interval. For the other nodes with larger populations, the Hoeffding approach gave much better estimations.

Figure 6 shows the results of using the CLT approach on the same three leaf nodes. Again, for the node with a small population, almost all the runs gave an estimation with a large confidence interval, since the CLT approach assumes a large population. On the other hand, this approach gave better results for the other nodes with large populations.

Combining both approaches for the case where the query node is a leaf node, we see that they behave very well in estimating average values for large populations, with CLT being slightly better. The Hoeffding approach is more conservative giving larger confidence intervals. For small populations, CLT should not be used while the Hoeffding approach may give some tolerable results.

6.2 AVG on Internal Node

We evaluated the adaptive-sampling algorithm described in Section 3.2 on an internal node with two leaf nodes of the Amazon data set. The first node had 3396 books, with an average price 48, and a standard deviation 58.38. The second node had 2157 books, with an average 21, and a standard deviation 27.92. The average of the internal node was 38. The total number of resources was 1000, and in each iteration we assigned $n = 200$ resources. We also implemented a naive algorithm that allocates the same number of resources to each leaf. Table 1 shows the results of the two approaches. The adaptive approach gave more accurate estimations of the average. Even

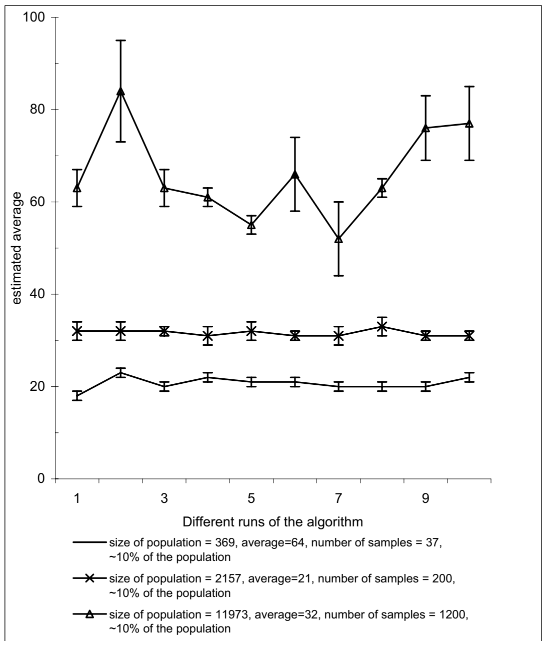

Figure 5: Hoeffding approach to answering an AVG query on a leaf node with a 95% confidence.

though the naive approach gave estimations with a small confidence interval, their estimations were far from the real average value.

Run	Estimated (Adaptive)	Estimated (Naive)
1	35 ± 4	31 ± 2
2	37 ± 4	30 ± 2
3	36 ± 5	29 ± 2
4	37 ± 4	30 ± 2
5	36 ± 6	31 ± 3
6	36 ± 5	35 ± 3
7	38 ± 4	31 ± 3
8	36 ± 5	34 ± 3
9	37 ± 3	32 ± 3
10	37 ± 3	32 ± 3

Table 1: Answering an AVG query on an internal node (real data). The real average is 38.

We also evaluated both approaches on the synthetic data, and a query was posted on an internal node with two leaf nodes. The first node had 3396 objects following a normal distribution, with an average 183 and standard deviation 59. The second node had 2157 objects following a normal distribution, with an average 87 and standard deviation 28. The average of the parent was 145. The results of the two approaches are shown in Table 2. Again, the adaptive approach gave much better estimations of the real average than the naive one.

6.3 MAX Queries

We implemented the algorithms in Section 4 for answering a MAX query. For the real data set, we first considered a node of 3396 books, with an average 48 and standard deviation 58. The

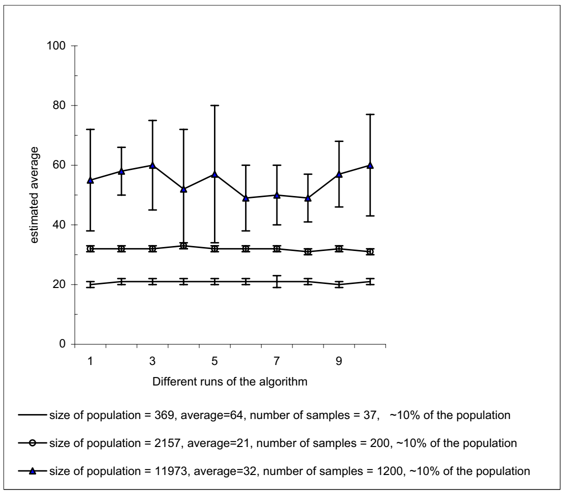

Figure 6: CLT approach to answering an AVG query on a leaf node with a 95% confidence.

Run	Est. AVG (Adaptive)	Est. AVG (Naive)
1	137 ± 6	135 ± 8
2	134 ± 6	131 ± 7
3	140 ± 5	134 ± 6
4	140 ± 7	134 ± 7
5	135 ± 6	128 ± 7
6	141 ± 6	132 ± 8
7	138 ± 8	135 ± 8
8	140 ± 6	137 ± 6
9	138 ± 8	134 ± 7
10	137 ± 7	133 ± 8

Table 2: Answering an AVG query on an internal node (synthetic data). The real average is 145.

real maximum value was 735. Table 3 shows how the adaptive algorithm presented in Section 4.1 behaved on this node. It shows that after 200 resources were allocated, the algorithm obtained an estimated maximum value with a high quantile. It found the real maximum value after 1600 resources were allocated.

Figure 7 shows how the adaptive algorithm and the naive algorithm behaved on the internal node (with two children) as described in Section 6.2. We had 1000 resources allocated in 5 iterations (200 each). The real maximum value was 1930. The results show that after the third iteration, the adaptive algorithm already found the real maximum. For the naive approach, even after the fifth iteration, its found maximum was still far from the real maximum.

6.4 Multiple Queries

We applied the ϵ-shake algorithm on a synthetic data set with the hierarchy and the four AVG queries Q_1, Q_2, Q_3 and Q_4 in Figure 4. We allocated 1600 resources in each iteration, and set $\epsilon = 0.00001$. Here we explained how the algorithm behaved for this data set. Leaves a, b, and c contribute only to Q_1, leaves d and e only to Q_4, leaf f to Q_2 and Q_4, and leaves g and h to Q_2, Q_3, and Q_4. To begin with, we get a presample assigning the same number of resources to each

Allocated Resources	Estimated Max	Quantile
50	165	0.852
100	500	0.974
200	335	0.974
400	355	0.976
600	557	0.987
800	575	0.991
1200	443	0.986
1400	500	0.988
1600	575	0.990
1800	735	0.994
2000	735	0.994

Table 3: Answering a MAX query on a leaf node (real data).

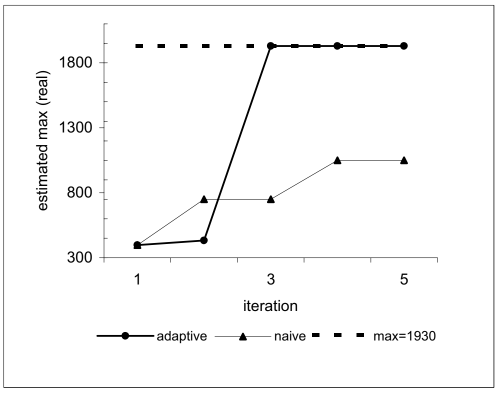

Figure 7: Adaptive versus naive algorithm for answering a MAX query on an internal node (real data)

leaf node, and calculate the first estimations of the average values for leaves and queries, as shown in Figure 8.

We compute the weight for each leaf. For leaf a, we have $w_a^2 = w_a^3 = w_a^4 = 0$, since this leaf does not contribute to queries Q_2, Q_3, and Q_4. For Q_1 we have $w_a^1{=}1 - \frac{\delta_\epsilon Q_{1a}}{\delta Q_1}{=}1 - \frac{\sqrt{(\frac{\theta f}{\theta a}(\sigma_a - \epsilon))^2 + (\frac{\theta f}{\theta b}\sigma_b)^2 + (\frac{\theta f}{\theta c}\sigma_c)^2}}{\sqrt{(\frac{\theta f}{\theta a}\sigma_a)^2 + (\frac{\theta f}{\theta b}\sigma_b)^2 + (\frac{\theta f}{\theta c}\sigma_c)^2}} = 1.9 \cdot 10^{-5}$. Summing all the weights we get $w_a = \sum_i w_a^i = 1.9 \cdot 10^{-5}$.

For leaf g we have $w_g^1{=}0$, $w_g^2 = 1.9 \cdot 10^{-5}$, $w_g^3 = 2.2 \cdot 10^{-5}$, $w_g^4 = 5.6 \cdot 10^{-5}$, and $w_g{=}\ w_g^1 + w_g^2 + w_g^3 + w_g^4 = 5.6 \cdot 10^{-5}$. In the same way we compute the weights for other leaf nodes. The total weight of all leaves of all the queries is $W = w_a + w_b + \ldots + w_h = 5.6 \cdot 10^{-4}$. Since we want to allocate 1600 resources in this iteration, the number of resources we assign to leaf a is $r_a = 1600 \cdot \frac{w_a}{W}$. Similarly we decide the number of resources to each other node. Figure 8 shows the resource allocations in the first two iterations.

The allocated resources show that the ϵ-shake algorithm captures two important factors. First, a leaf with a high estimated error in an iteration receives more resources in the next iteration in order to improve its estimation. For instance, nodes a and b are both relevant to query Q_1. In the first iteration, they have the same contribution to the query, but they receive different numbers (204 and 59) of resources in the next iteration due to the difference in their estimated errors. Second, leaves interrelated with many queries receive more resources than others, since every query

First Iteration

Query Estimations
$Q_1 = 43.9 \pm 1.2$
$Q_2 = 41.6 \pm 1.5$
$Q_3 = 38.7 \pm 2.0$
$Q_4 = 35.1 \pm 1.0$

Leaves	**Weights**	**Allocated Resources**
$a = 51.0 \pm 2.4$	$w_a = 1.9 \cdot 10^{-5}$	$r_a = 204$
$b = 33.5 \pm 0.7$	$w_b = 5.5 \cdot 10^{-6}$	$r_b = 59$
$c = 47.2 \pm 2.5$	$w_c = 2.0 \cdot 10^{-5}$	$r_c = 214$
$d = 28.7 \pm 0.7$	$w_d = 2.8 \cdot 10^{-6}$	$r_d = 31$
$e = 22.2 \pm 2.1$	$w_e = 8.2 \cdot 10^{-6}$	$r_e = 88$
$f = 47.3 \pm 2.0$	$w_f = 1.7 \cdot 10^{-5}$	$r_f = 185$
$g = 34.8 \pm 3.8$	$w_g = 5.6 \cdot 10^{-5}$	$r_g = 600$
$h = 42.7 \pm 1.4$	$w_h = 2.1 \cdot 10^{-5}$	$r_h = 223$

Second Iteration

Query Estimations
$Q_1 = 44.0 \pm 1.0$
$Q_2 = 40.6 \pm 0.5$
$Q_3 = 38.4 \pm 0.5$
$Q_4 = 36.4 \pm 0.4$

Leaves	**Weights**	**Allocated Resources**
$a = 49.9 \pm 2.0$	$w_a = 2.3 \cdot 10^{-5}$	$r_a = 82$
$b = 31.7 \pm 0.5$	$w_b = 4.8 \cdot 10^{-6}$	$r_b = 18$
$c = 47.0 \pm 1.7$	$w_c = 1.7 \cdot 10^{-5}$	$r_c = 62$
$d = 28.7 \pm 0.7$	$w_d = 1.1 \cdot 10^{-5}$	$r_d = 38$
$e = 19.1 \pm 0.7$	$w_e = 6.1 \cdot 10^{-6}$	$r_e = 22$
$f = 46.4 \pm 1.2$	$w_f = 5.7 \cdot 10^{-5}$	$r_f = 204$
$g = 29.5 \pm 0.7$	$w_g = 1.1 \cdot 10^{-4}$	$r_g = 396$
$h = 45.6 \pm 0.7$	$w_h = 2.2 \cdot 10^{-4}$	$r_h = 781$

Figure 8: First and second iterations of the ϵ-shake algorithm on the queries in Figure 4.

adds some weight to them. For example, in the second iteration, although leaf a has a bigger estimated error than g, the latter receives more resources (600) than the former (204), since the latter contributes to three queries while the former only to one.

7 Conclusions

In this paper we studied the problem of answering aggregation queries on data published in hierarchical Web structures. We proposed solutions that approximate the answer to such a query by sampling, assuming that we are given limited access resources. Our adaptive sampling algorithms take into account the fact that in a hierarchical structure, different leaves have different distributions and contributions to the accuracy of the answer to the query, and such information is not known a priori. We studied various cases of the problem, and developed corresponding algorithms.

We have conducted experiments to evaluate these algorithms.

Many open problems need to be further studied. For instance, we can develop more effective algorithms if some information about the distributions of the leaf nodes are known. We need more sophisticated algorithms when the data distributions of leaf nodes are skewed, especially if the sizes of the populations on the leaves are very different.

References

[1] S. Chaudhuri, G. Das, R. Motwani, and V. Narasayya. Overcoming limitations of sampling for aggregation queries. In *ICDE*, 2001.

[2] J. Czyzowicz, E. Kranakis, D. Krizanc, A. Pelc, and M. V. Martin. Evaluation of hotlink assignment heuristics for improving web access. *Computing (IC' 01)*, 2:793–799, June 2001.

[3] M. Garofalakis, J. Gehrke, and R. Rastogi. Querying and mining data streams: You only get one look. In *ACM SIGMOD*, 2002.

[4] P. J. Haas, J. F. Naughton, and A. N. Swami. On the relative cost of sampling for join selectivity estimation. In *PODS*, pages 14–24, 1994.

[5] B. Heeringa and M. Adler. Optimal website design with the constrained subtree selection problem. In *31st Intern. Colloquium on Automata, Languages, and Programming (ICALP)*, pages 436–447, 2004.

[6] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. In *SIGMOD Conference*, pages 171–182, 1997.

[7] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58:13–30, 1963.

[8] R. J. Lipton and J. F. Naughton. Query size estimation by adaptive sampling (extended abstract). In *PODS*, pages 40–46, 1990.

[9] R. J. Lipton, J. F. Naughton, D. A. Schneider, and S. Seshadri. Efficient sampling strategies for relational database operations. In *Selected papers of the 4th international conference on Database theory*, pages 195–226, 1993.

[10] R. J. Lipton and D. A. Schneider. Practical selectivity estimation through adaptive sampling. In *ACM SIGMOD*, pages 1–11, 1990.

[11] F. Olken. Random sampling from databases - bibliography, http://pueblo.lbl.gov/~olken/mendel/sampling/bibliography.html.