

# A Bayesian Method for Guessing the Extreme Values in a Data Set\*

Mingxi Wu  
Department of Computer and Information  
Sciences and Engineering  
University of Florida  
Gainesville, FL, USA, 32611  
mwu@cise.ufl.edu

Christopher Jermaine  
Department of Computer and Information  
Sciences and Engineering  
University of Florida  
Gainesville, FL, USA, 32611  
cjermain@cise.ufl.edu

## ABSTRACT

For a large number of data management problems, it would be very useful to be able to obtain a few samples from a data set, and to use the samples to guess the largest (or smallest) value in the entire data set. Min/max online aggregation, top- $k$  query processing, outlier detection, and distance join are just a few possible applications. This paper details a statistically rigorous, Bayesian approach to attacking this problem. Just as importantly, we demonstrate the utility of our approach by showing how it can be applied to two specific problems that arise in the context of data management.

## 1. INTRODUCTION

This paper deals with a ubiquitous problem in data management: guessing the maximum/minimum value (or some other extreme statistics) over a data set. Stated simply, we wish to guess the  $k^{\text{th}}$  largest  $f(d)$  value over all  $d \in D$  for a data set  $D$ . More formally, given an arbitrary function  $f()$ , our goal is to accurately guess the  $k^{\text{th}}$  largest  $f()$  value  $f_{(k)}$  for all  $d \in D$ , such that  $|\{f(d) : f(d) > f_{(k)} \wedge d \in D\}| = k - 1$ .

This particular problem arises in many applications, for example:

- As pointed out by Donjerkovic and Ramakrishnan [6], in top- $k$  query processing knowing the cutoff value beforehand allows the “top- $k$ ” portion of the query to be transformed into a relational selection. The resulting query can then be processed without modification to the database engine.
- In outlier detection for data mining, the state-of-the-art algorithm [2] prunes points from the outlier candidate set when it has been determined that there are too many points that are close to the candidate. If it were possible to accurately guess the distance to a point’s  $k^{\text{th}}$  nearest neighbor, this pruning could be done without actually finding those close-by points.
- In distance join processing [9, 18], the goal is to find the  $k$  closest pairs over two different data sets. The fact that  $k$  is

\*Material in this paper is based upon research funded by the National Science Foundation under grant no. 0612170.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, to post on servers or to redistribute to lists, requires a fee and/or special permission from the publisher, ACM.

VLDB '07, September 23-28, 2007, Vienna, Austria.

Copyright 2007 VLDB Endowment, ACM 978-1-59593-649-3/07/09.

supplied to the join (as opposed to a cutoff distance) makes the query more useful, but it greatly complicates the computation. If the cutoff distance were known beforehand, then the problem can be solved using any one of a large number of efficient algorithms [1, 14, 5].

Many other applications exist, though space precludes listing them all here.

Under certain circumstances, guessing  $f_{(k)}$  is trivial. If  $f(d)$  simply returns an attribute of  $d$ , then the solution can be as easy as pre-computing and storing the largest  $k$  attribute values from the data set. However, the problem can become arbitrarily difficult depending on the nature of  $f()$ . In the general case,  $f()$  may encode an unanticipated, arbitrary multi-attribute relational selection predicate – that is,  $f(d)$  returns  $-\infty$  if some selection predicate does not evaluate to *true*. In other cases,  $f()$  may perform an arbitrary, non-linear numerical computation over the attributes of  $d$  that is impossible to anticipate.

## A Bayesian Approach

In this paper, we propose a novel approach to solving this problem. Since we are trying to guess extreme values for any arbitrary and unanticipated function  $f()$ , we argue that it is impossible to solve the problem by using a statistical synopsis to model the data. Models for the data are often useful for describing what is “typical”, but the extreme value queries we are interested in specifically refer to the outliers in the data. Thus, standard approaches are of limited utility. For example, consider a 1% sample of a database having 100 million records where we are interested in  $f_{(10)}$ , but we have no prior knowledge of  $f()$  and so it is not possible to bias the sample to larger  $f()$  values. Since we have less than a 10% chance of sampling any of the records resulting in one of the 10 largest  $f()$  values, how can we guess  $f_{(10)}$  with high accuracy?

Thus, rather than trying to guess  $f_{(k)}$  by modeling the data, we instead guess  $f_{(k)}$  by watching and modeling the behavior of the queries we have seen.<sup>1</sup>

Some query aggregate functions may result in  $f()$  values that are typically very small, with a few tremendous outliers that greatly boost the value of  $f_{(k)}$ . Some query aggregate functions may result in  $f()$  values that are tightly, normally distributed around the mean  $f()$  value, meaning that  $f_{(k)}$  is not too different from the typical  $f()$  value. As queries are asked, our method watches and learns what “typical” queries tend to look like. Then, when a new query is asked, we look at the first few  $f()$  values obtained and decide (in

<sup>1</sup>We will use the term *query* very loosely in the paper, and its exact meaning will depend upon the application.

a statistically meaningful way) which type of the “typical” queries we are experiencing.

Of course, we may be wrong when we guess what type of query has been asked, and so there is a degree of uncertainty with our belief. This uncertainty is incorporated into the probabilistic guarantee on the accuracy of our guessed  $f_{(k)}$ . In this way, our method is an example of a so-called *Bayesian* statistical technique, in that we make use of a “prior” or guessed query distribution in order to associate a belief with the type of query that has been issued.

## Our Contributions

This paper has the following technical contributions:

- We propose a new method for guessing the answer to an extreme value query over an arbitrary function. The method is statistically rigorous and makes use of unique Bayesian statistical techniques. Such techniques have been mostly ignored in the data management literature to date.
- Along with the approximate answer, our method returns the distribution associated with the guess’ error, and so it can be used to associate confidence bounds on the guess’ accuracy.
- We devise a method to learn the prior query distribution by watching query results as they are produced. The learning algorithm requires that for each query result, we compute only three aggregate values. In this way, the learning method is inexpensive, and can easily be incorporated into a DBMS or a specific application with little or no cost.
- We argue for the utility of our approach by detailing two separate applications of our method to specific problems that occur when dealing with large databases.

## 2. PROBLEM DEFINITION

In this section, we formalize our problem and describe the solution that we study in the paper.

### 2.1 Estimating the Extreme Value

Our goal is to provide estimation algorithms that will support processing for extreme value queries of the form:

```
SELECT g1(d1)
FROM D AS d1
WHERE g2(d1) AND k - 1 = (
    SELECT COUNT(*)
    FROM D AS d2
    WHERE g2(d2) AND g1(d1) < g1(d2);
```

In the above query,  $g_1()$  is an arbitrary function that maps a tuple  $d$  to a real value<sup>2</sup> and  $g_2()$  is a relational selection predicate. This query asks for the  $k^{th}$  largest  $g_1(d)$  value over all the database tuples that satisfy the selection predicate  $g_2()$ . If we change “ $g_1(d_1) < g_1(d_2)$ ” to “ $g_1(d_1) > g_1(d_2)$ ”, then the query is easily modified to ask for the  $k^{th}$  smallest  $g_1(d)$  value when the selection predicate defined by  $g_2()$  is satisfied. For ease of exposition, in the remainder of the paper we assume that the search is for a large value.

The fact that we have two separate functions  $g_1()$  and  $g_2()$  complicates things a bit, so we encode both individual functions within a single function  $f()$ :

$$f(d) = \begin{cases} g_1(d) & \text{if } g_2(d) \text{ is true} \\ -\infty & \text{otherwise} \end{cases}$$

<sup>2</sup>For ease of exposition, we assume that each tuple maps to a distinct real value.

We use the notation  $f_{(k)}$  to denote the answer to the query.

## 2.2 A Natural Estimator

Assuming that there are no index structures to help us locate tuples with specific  $f()$  values, an obvious solution to the problem is to sequentially scan the database once and evaluate  $f()$  over each tuple. Then we can return the  $k^{th}$  largest  $f()$  value encountered. However, this algorithm may be too slow if the database is large, or if  $f_{(k)}$  must be evaluated repeatedly for different functions (as in the application to outlier detection that we will consider).

The fundamental idea in this paper is that, in order to obtain a sub-linear-speed algorithm to compute  $f_{(k)}$ , one can use a simple random sample (without replacement) from the database. By examining the  $f()$  values in this sample, it may be possible to estimate the  $k^{th}$  largest/smallest value in the query result set.<sup>3</sup> The estimator that we study works as follows. Suppose that we have access to  $n$  samples from a database of size  $N$ . Then a natural estimator for the  $k^{th}$  largest value in the query is the  $(k')$ <sup>th</sup> largest value in the sample, where  $\frac{k'}{n} = \frac{k}{N}$ . Since to make use of this estimator,  $k'$  must be an integer, we use  $k' = \lceil \frac{n}{N} \times k \rceil$ . In the remainder of the paper, we use  $\widehat{f_{(k)}}$  to denote the  $(k')$ <sup>th</sup> largest value in the sample.

For example, suppose that we wish to guess the largest value in a database of size 15. Applying  $f()$  to each tuple, we obtain the set:

$$\{1, 2, -\infty, 4, 5, -\infty, 7, 8, 9, -\infty, 11, -\infty, -\infty, 14, 15\}$$

Thus,  $f_{(1)} = 15$ . Now, we take a five-item sample from this set, which happens to be:

$$\{11, -\infty, 4, 1, -\infty\}$$

Then our estimator  $\widehat{f_{(1)}}$  is 11, where  $k' = \lceil \frac{5}{15} \times 1 \rceil$ .

Characterizing the accuracy of this estimator is far from trivial. The fundamental question we ask in this paper is:

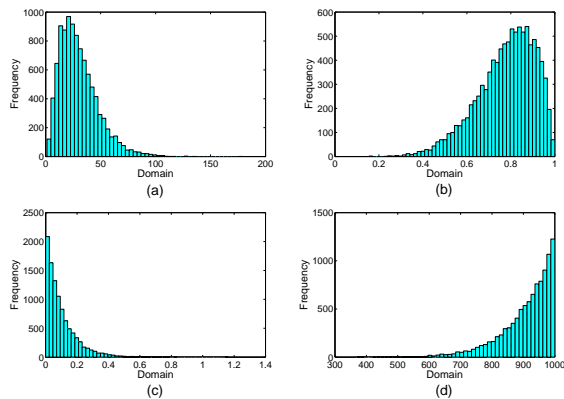
**How does the estimator  $\widehat{f_{(k)}}$  relate to the true  $f_{(k)}$ ?**

Given a rigorous characterization of this relationship, it is possible to both correct  $\widehat{f_{(k)}}$  to obtain an even better estimate, and to characterize the accuracy of the estimator. Since the difference between  $\widehat{f_{(k)}}$  and  $f_{(k)}$  is affected by the scale of the values under consideration, this paper studies the behavior of the ratio  $\frac{\widehat{f_{(k)}}}{f_{(k)}}$  as a way to characterize the accuracy of the estimator. In particular, we are interested in characterizing the *distribution* of this ratio, because this distribution facilitates the use of  $\widehat{f_{(k)}}$  to produce confidence bounds on  $f_{(k)}$ . For example, if we know that there is a 90% chance that the ratio is between  $l$  and  $h$ , then there is a 90% chance that  $f_{(k)}$  is between  $l \times \widehat{f_{(k)}}$  and  $h \times \widehat{f_{(k)}}$ .

## 3. OVERVIEW OF OUR APPROACH

This section gives an overview of the approach that we use to characterize the ratio  $\frac{\widehat{f_{(k)}}}{f_{(k)}}$ . First, we discuss the statistical property of the database most relevant to the characterization: the shape of the right tail of the distribution of  $f()$  values. Then we give an overview of a unique Bayesian approach to dealing with the importance of the tail’s shape.

<sup>3</sup>Although we consider the case where a single sample is used for a single estimate, the technique developed in this paper can easily be extended to deal with the online case, where the entire randomized database will be scanned. At each instant during the scan, the set of tuples retrieved thus far is a sample of the database.



**Figure 1: The histograms of four different query result sets with different domains(scales). (a) and (c) have long tails to the right. (b) and (d) have no tails to the right.**

### 3.1 Importance of the Query Shape

Unlike many other estimation problems, characterizing the estimator  $\widehat{f}_{(k)}$  is extremely challenging, because unlike classical estimation problems where simple statistical properties such as the distribution’s variance are important, the actual shape of the query result set’s distribution is most closely related to the accuracy of the estimator. This is best illustrated by an example.

Figure 1 depicts a set of histograms showing the distributions of four synthetic query result sets. Each query result set contains 10,000 values. Now, imagine that we wish to use a sample of size 100 to compute  $\widehat{f}_{(1)}$ , and we ask: How accurate is  $\widehat{f}_{(1)}$  as an estimate for  $f_{(1)}$ ? To answer this question, we re-sample (without replacement) 100 times to produce 100 different  $\widehat{f}_{(1)}$  values, and compute the median for  $\frac{f_{(1)}}{\widehat{f}_{(1)}}$  over each data set. The median values recorded are 3.07, 1.06, 4.45 and 1.01 for (a), (b), (c), and (d), respectively.

The relative magnitudes of these four values can be explained by examining Figure 1. The distributions corresponding to queries (a) and (c) have long tails to the right, so one has only a small chance of sampling any values close to the tip of the right tail where  $f_{(1)}$  is located. Therefore, we observe a large ratio between the true answer and the estimator. In contrast, the shapes of query (b) and (d) have no tail to the right, so one would expect that a sample has an excellent chance of including values close to  $f_{(1)}$ . As a result, we observe a ratio close to one for these two queries.

### 3.2 Basics of the Bayesian Approach

Since the query shape is so important when evaluating the accuracy of  $\widehat{f}_{(k)}$ , it must be incorporated into the process of characterizing the ratio  $\frac{f_{(k)}}{\widehat{f}_{(k)}}$ . Our basic approach is to assume that there exists a large number of possible query shapes: from “easy” shapes with no skew to very “hard” shapes with a heavy right skew. Each shape has a weight or probability associated with it that specifies the extent to which we think this is the current shape we are experiencing – representing such a belief with a probability is the hallmark of the so-called “Bayesian” statistical approach [13].

The initial set of weights that we start out with before any data have been encountered are known using Bayesian terminology as the *prior distribution*. While there are many ways to develop a reasonable prior distribution, we choose to learn the prior from the

historical query workload. Then, as data from a particular query are encountered, the weights are updated in a statistically rigorous fashion to take into account the new data. In Bayesian terminology, the updated weights represent the *posterior distribution*. These updated weights are then used to produce confidence bounds. For example, if a database sample of reasonable size is obtained that is consistent with a query having a heavy rightward skew, the updated weights will tend to favor query shapes with corresponding rightward skew, and the confidence bounds for the accuracy of  $\widehat{f}_{(k)}$  that we report will be suitably wide.

### The “Dangers” of the Bayesian Approach

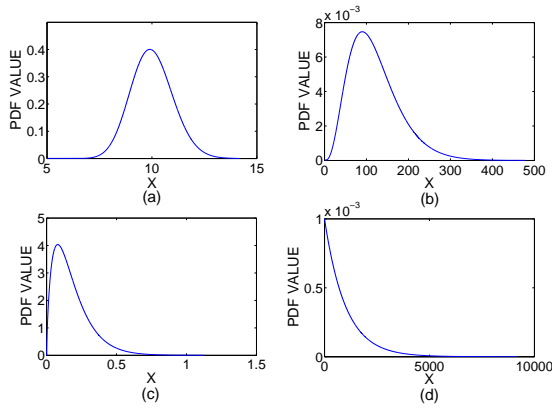
At first glance, assuming the existence of a prior distribution may seem dangerous. Since we will learn our prior from the previously observed queries, we are assuming that a new query will never be totally different from all of the queries in the training workload. In the case where we see a “new” query, the shape corresponding to the new query will necessarily have a zero prior weight, since the query was totally unanticipated. If the new query has a tail that is far nastier than anything else we have ever seen, then we may be too aggressive with our confidence bounds – this is the danger inherent in the Bayesian approach.

In fact, related dangers are inherent in *all* estimation techniques that do not have access to all of the data, including classic methods that are widely used in the data management literature. For example, consider the classical, sampling-based estimator for a SUM SQL query [8, 7]: first a  $1/\alpha$  sample of the database is taken, then the query is applied to the sample and the result is scaled up by a factor of  $\alpha$ . It is an often-ignored fact that in order to bound the accuracy of such an estimate in the classical fashion, *the variance of the estimator is also estimated from the same sample*. If the variance estimate is too low (which may be the case if there is one particularly high-value record that did not appear in the sample) then any resulting confidence bounds are worthless. The implicit assumption underlying the classic method is that the database characteristic in question – the variance – can be estimated accurately from the sample. In comparison, the Bayesian approach makes an explicit assumption regarding the availability of a prior distribution. In either case, the possibility of an error exists. In fact, a statistician from the so-called “Bayesian” school would argue that it is better to make such assumptions explicit using a prior than to hide behind arguments such as the unbiasedness of a variance estimate. As we will show experimentally, our application of the Bayesian approach is very robust to errors in the prior, and turns out to be quite successful in practice.

### 3.3 Proposed Bayesian Inference Framework

Given this background, we now describe the three steps of our Bayesian inference framework:

1. The *learning phase* uses statistical methods to build a prior shape model composed of a number of candidate shape patterns. Each shape pattern represents a class of queries. A weight is assigned to each shape pattern, indicating how likely a future query’s shape matches that shape pattern. Both the weights and the shape patterns are learned offline, from the historical query workload using an EM algorithm [4].
2. The *characterization phase* derives an error distribution for each learned shape pattern. Before we can use the learned prior shape model to predict the behavior of  $\widehat{f}_{(k)}$  on a real-life query, as a preparation for the next phase we need to derive the distribution of  $\frac{f_{(k)}}{\widehat{f}_{(k)}}$  for each learned shape pattern. This is



**Figure 2: The PDFs of four Gamma distributions with increasing skew to the right from (a) to (d).**

done using Monte Carlo methods [17], after the shape model has been learned but before it is time to actually answer an extreme-value query.

3. The *inference phase* uses the results of the characterization phase to produce the error distribution for an actual query. The characterization phase applies only to the learned shape patterns, and not to any real-life query. When it is time to actually answer a query online, the prior weights of the shape model are updated based upon the observed samples to produce the posterior weights. Using the posterior weights and the error distribution for each shape pattern, an error distribution for the ratio  $\frac{f(k)}{\widehat{f(k)}}$  is obtained. Since  $\widehat{f(k)}$  can be computed from the sample, confidence bounds on  $f(k)$  can easily be derived from the resulting distribution.

The next three sections of the paper describe each of the three phases in more detail. In these sections, we simplify the exposition by assuming  $f()$  never returns  $-\infty$ ; that is, the size of the query result set is the database size. In the Experiments Section where the framework is actually applied to two specific problems, we will discuss how to remove this assumption when necessary.

## 4. THE LEARNING PHASE

Any Bayesian method requires a generative, probabilistic model for the data. The process should be both general (in the sense that it allows the production of any data set that might be observed) and specific (in the sense that it produces all important properties of the underlying data and is tailored for the specific problem at hand).

In our case, each individual “data point” that is produced by the generative process is a single query result set. Since (as described in Section 3.1) the shape of the query result set is so important in determining the quality of the estimator  $\widehat{f(k)}$ , the generative model will pay special attention to how the shape is handled.

Informally, we assume the following generative process for producing each query result set:

- First, a biased die is rolled to determine by which shape pattern the query result set will be generated. We assume the existence of some set of  $c$  different shape patterns, and the die roll selects one of them.
- Next, an arbitrary scale for the query is randomly selected. This scale defines the magnitude of the items in the query

result set. In Figure 1, the scale determines how large the labels are on the  $X$ -axis.

- Finally, the shape and scale are used as inputs to instantiate a parametric model for the data. For reasons described subsequently, we will make use of the Gamma distribution from statistics as our parametric model. This distribution is repeatedly sampled from to produce the query result set.

Given the intuitive process described above, the next step is to formalize it. Mathematically, this is done by defining a *probability density function* (PDF) for the process. This is a function that takes as an input a query result set, and returns how probable it is that this query result set would be produced by the process. After defining the PDF, we will then consider how to “learn” the model; that is, we consider how to tailor the model to a specific query workload.

### 4.1 Choosing an Appropriate Parametric Model

It is first necessary to choose some parametric distribution to model the query shape. Given the discussion of Section 3.1, there is one overarching concern: our distribution must be able to model arbitrarily long tails to the right. Modeling all of the dips and bumps of a real-life distribution is not necessary, because we are concerned only with the relationship between the distribution’s tail and the rest of its mass.

Given this consideration, the Gamma distribution family becomes a natural choice, since it can produce a shape with arbitrary right-leaning skew. Figure 2 shows the PDFs of four instances of the Gamma family, each with increasing skew and longer tails to the right. Figure 2(a) depicts a bell (normal) shape, which does not have any skew to the right and only a very short tail relative to the distribution’s variance, whereas Figure 2(d) is highly skewed to the right with an exceedingly long tail.

We stress that though the Gamma distribution underlies our model, we do *not* assume that the  $f()$  values in the database look anything like a Gamma distribution. The Gamma distribution is used only to model the relationship between the values in the far right tail of the data distribution and the values that are more likely to be sampled – those that are closer to the main body of the distribution. The Gamma does this well because of its ability to take on shapes having arbitrary skew. As we will show experimentally, using the Gamma distribution, our method can handle data sets that could not possibly have been sampled from a Gamma distribution, including those with a left skew and those with multiple modes, including very small modes or “bumps” far out in the right tail.

### 4.2 Deriving the PDF

We now turn our attention to deriving the PDF associated with the resulting, three-step, generative process. Formally, the PDF of the Gamma distribution can be expressed in terms of the Gamma function  $\Gamma^4$ :

$$p(x|\alpha, \beta) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x}, \quad x > 0 \quad (1)$$

In Equation 1, the parameter  $\alpha > 0$  is known as the *shape parameter*, since it influences the shape or skew of the distribution, while the parameter  $\beta > 0$  is called the *inverse scale parameter*, since  $\frac{1}{\beta}$  influences the domain (scale) of the distribution.

Let  $\vec{y} = \langle d_1, \dots, d_N \rangle$  denote a query result set which has  $N$  matching tuples. Assuming that the tuples are independently drawn

<sup>4</sup>The Gamma function is defined as  $\Gamma(\alpha) = \int_0^\infty t^{\alpha-1} e^{-t} dt$ , where  $\alpha > 0$ .

from a Gamma distribution, using Equation 1 the likelihood of observing a query  $\vec{y}$  given  $\alpha$  and  $\beta$  is:

$$\begin{aligned} p(\vec{y}|\alpha, \beta) &= \prod_{i=1}^N \left\{ \frac{\beta^\alpha}{\Gamma(\alpha)} d_i^{\alpha-1} e^{-\beta d_i} \right\} \\ &= \frac{\beta^{N\alpha}}{\Gamma^N(\alpha)} M^{\alpha-1} e^{-\beta S} \end{aligned} \quad (2)$$

In Equation 2,  $M = \prod_{i=1}^N d_i$  and  $S = \sum_{i=1}^N d_i$ .

Since we are interested in characterizing a ratio, we are uninterested in the scale parameter and do not want to bias our model to any particular scale. Thus, we treat the inverse scale parameter  $\beta$  as a random variable that is uniformly chosen from  $(0, max)$ , where  $max$  is a huge number chosen to be large enough that it permits a scale that is arbitrarily small. Then, the likelihood of observing a query  $\vec{y}$  given the shape  $\alpha$  and unknown  $\beta$  is given by:

$$\begin{aligned} p(\vec{y}|\alpha) &= \int_0^{max} p(\beta) \times \frac{\beta^{N\alpha}}{\Gamma^N(\alpha)} M^{\alpha-1} e^{-\beta S} d\beta \\ &\approx \frac{1}{max} \times \frac{M^{\alpha-1}}{\Gamma^N(\alpha)} S^{-N\alpha-1} \Gamma(N\alpha + 1) \end{aligned} \quad (3)$$

Equation 3 is the result of taking expectation of Equation 2 with respect to  $\beta$ . It shows that evaluating the likelihood of a given query result set requires exactly three aggregate values:  $M$ ,  $S$  and  $N$ , denoting the product and the sum of all the values in the query's result set and the size of the query, respectively. Consequently, these three numbers are all that we need to collect with respect to each query. Subsequently,  $\vec{y}$  will refer to this triplet  $\langle M, S, N \rangle$ .

Note that Equation 3 is valid for a given shape parameter. Since our model assumes that a shape parameter is chosen at random from a weighted set where the probability of choosing shape  $\alpha_j$  is  $w_j$ , the likelihood of observing an entire query set  $\vec{y}$  is:

$$f^*(\vec{y}|\Theta) = \sum_{j=1}^c w_j p(\vec{y}|\alpha_j) \quad (4)$$

In Equation 4, the  $w_j$ s are each non-negative weights satisfying the constraint that  $\sum_{j=1}^c w_j = 1$ . The complete set of model parameters is  $\Theta = \{\theta_1, \dots, \theta_c\}$ , where  $\theta_j = \{w_j, \alpha_j\}$ .

### 4.3 Learning the Parameters

$\Theta$  is unknown and must be learned from the historical workload. To learn  $\Theta$ , we follow the basic principle of *Maximum Likelihood Estimation* (MLE), whose goal is to find the parameter set most likely to have produced the observed data.

Given a set of independent, historical queries  $Y = \{\vec{y}_1, \dots, \vec{y}_r\}$ , applying Equation 4 the likelihood of observing  $Y$  is:

$$L(\Theta|Y) = \prod_{i=1}^r f^*(\vec{y}_i|\Theta)$$

Often, it is preferable to work with  $\log(L(\Theta|Y))$  because the product given above becomes a summation. That is, we wish to find  $\Theta^*$  so that it maximizes:

$$\Lambda = \underset{\Theta}{\operatorname{argmax}} \log(L(\Theta|Y))$$

In order to optimize the objective function  $\Lambda$ , we employ the *Expectation-Maximization (EM)* framework [4] from statistics and machine learning to iteratively maximize the log-likelihood.

### The EM Algorithm

EM is used to solve MLE problems made difficult by the fact that there are one or more ‘‘hidden’’ variables that cannot be observed in the data. EM is an iterative method, whose basic outline is described in Algorithm 1.

---

#### Algorithm 1 Basic EM algorithm

---

- 1: **while** The model continues to improve **do**
  - 2: Let  $\Theta$  be the current ‘‘best guess’’ as to the optimal configuration of the model
  - 3: Let  $\bar{\Theta}$  be the next ‘‘best guess’’ as to the optimal configuration of the model
  - 4: **E-Step:** Compute  $Q$ , the expected value of  $\Lambda$  with respect to all possible values of the hidden variables. The probability of observing each possible set of hidden values is computed using  $\Theta$ .
  - 5: **M-Step:** Choose  $\bar{\Theta}$  so as to maximize the value for  $Q$ .  $\bar{\Theta}$  then becomes the new ‘‘best guess’’.
  - 6: **end while**
- 

In our problem, the hidden variables are the identities of each particular shape that was used to produce each training query. Since the details of some derivations below are similar to the example in [4], most of the resembling derivations are omitted. As a result of the **E-Step**, we have:

$$\begin{aligned} Q(\Theta, \bar{\Theta}) &= \sum_{j=1}^c \sum_{i=1}^r \log(\bar{w}_j) \times p(j|\vec{y}_i, \Theta) + \\ &\quad \sum_{j=1}^c \sum_{i=1}^r \log(p(\vec{y}_i|\bar{\alpha}_j)) \times p(j|\vec{y}_i, \Theta) \end{aligned} \quad (5)$$

In Equation 5,  $p(j|\vec{y}_i, \Theta)$  is the posterior probability of query  $\vec{y}_i$  coming from the  $j^{th}$  shape pattern's distribution, and is given by:

$$p(j|\vec{y}_i, \Theta) = \frac{w_j p(\vec{y}_i|\alpha_j)}{\sum_{l=1}^c w_l p(\vec{y}_i|\alpha_l)}$$

In the **M-Step**, we need to obtain the update equations for the weights  $\bar{w}_j$  and the shapes  $\bar{\alpha}_j$ . To update the weights, we use a Lagrange multiplier to maximize  $Q$  with respect to  $\bar{w}_j$ . This gives us the following update equation for  $w_j$ :

$$\bar{w}_j = \frac{1}{r} \sum_{i=1}^r p(j|\vec{y}_i, \Theta)$$

Next we maximize  $Q$  with respect to each  $\bar{\alpha}_j$  by taking derivative of  $Q$  and setting the result to zero. The part of  $Q$  relevant to  $\bar{\alpha}_j$  is:

$$Q_2 = \sum_{j=1}^c \sum_{i=1}^r \log(p(\vec{y}_i|\bar{\alpha}_j)) \times p(j|\vec{y}_i, \Theta)$$

Unfolding the log operation and taking the derivative with respect to  $\bar{\alpha}_j$ , we have:

$$\begin{aligned} \frac{\partial Q_2}{\partial \bar{\alpha}_j} &= \sum_{i=1}^r \{ \log M_i - N_i \psi(\bar{\alpha}_j) - N_i \log S_i + \\ &\quad N_i \psi(N_i \bar{\alpha}_j + 1) \} \times p(j|\vec{y}_i, \Theta) \end{aligned} \quad (6)$$

In Equation 6,  $\psi(\cdot)$  is the Digamma function<sup>5</sup>. To update  $\bar{\alpha}_j$  we set Equation 6 to zero and solve it by the bisection method [3].

---

<sup>5</sup>The Digamma function is defined as  $\psi(z) = \frac{\Gamma'(z)}{\Gamma(z)}$ .

The EM algorithm repeatedly applies these update equations in an iterative fashion until the parameters begin to stabilize (this is typically measured by an iteration-to-iteration fractional change in  $\Theta$  that is less than 1%).

## 5. THE CHARACTERIZATION PHASE

The learning phase provides us with a set of weighted shape patterns that describe the historical workload. As a preparation for the inference phase to make use of these shapes, we need to derive the error distribution associated with each shape. In this section, we show how to determine the distribution of the ratio  $\frac{\widehat{f}_{(k)}}{f_{(k)}}$  for a query result set that we know has been generated by a shape parameter  $\alpha$  and a scale parameter  $\frac{1}{\beta}$ . The extension to unknown parameters is considered in the next section.

Since a query is treated as a sample from a parametric distribution, our estimator  $\widehat{f}_{(k)}$  and the final answer  $f_{(k)}$  can be viewed as a result of the following two-stage sampling process:

- Stage One. A query is produced by drawing a sample of size  $N$  from the distribution  $\text{Gamma}(x|\alpha, \beta)$ . The  $k^{\text{th}}$  largest value in this sample is  $f_{(k)}$ .
- Stage Two. In order to estimate  $f_{(k)}$ , a subsample of size  $n$  is drawn without replacement from the sample obtained in stage one. The  $(k')^{\text{th}}$  largest value in this subsample is our estimator  $\widehat{f}_{(k)}$ .

Given this process, it is clear that  $f_{(k)}$  and  $\widehat{f}_{(k)}$  are correlated. As a result, it is very hard to analytically obtain the distribution of the ratio between them. Therefore, we resort to Monte Carlo methods to obtain their ratio's approximate distribution, expressed in the form of the cumulative distribution function (CDF).

---

### Algorithm 2 Naive Monte Carlo Sampling

---

**Input:**  $N, k, n, \alpha, \beta, num$

**Output:** the Monte Carlo sample array  $MC$

- 1: Let  $MC = \emptyset$
  - 2: **for**  $i = 1$  To  $num$  **do**
  - 3: Stage 1: draw an i.i.d. sample of size  $N$  from the  $\text{Gamma}(x|\alpha, \beta)$  distribution, then find  $f_{(k)}$  from it
  - 4: Stage 2: draw a subsample of size  $n$  from the sample obtained in stage 1, then find  $\widehat{f}_{(k)}$  from it
  - 5:  $MC = MC \cup \left\{ \frac{\widehat{f}_{(k)}}{f_{(k)}} \right\}$
  - 6: **end for**
  - 7: Return  $MC$
- 

### 5.1 Naive Monte Carlo Sampling

In order to obtain the distribution of a statistic, the Monte Carlo approach obtains a large number of independent samples of the statistic directly from the underlying generative model. The samples can then be organized into a sorted list so that the approximate CDF for the target distribution can be calculated by simply counting the fraction of samples less than the CDF's input variable.

It is not hard to imagine how a naive Monte Carlo algorithm for obtaining our particular error distribution would work. The algorithm would simply repeat the above two-stage process  $num$  times (where  $num$  is the number of Monte Carlo samples to produce), and return the array of  $num$  ratios produced. The basic Monte Carlo approach is given as Algorithm 2. The input parameters are

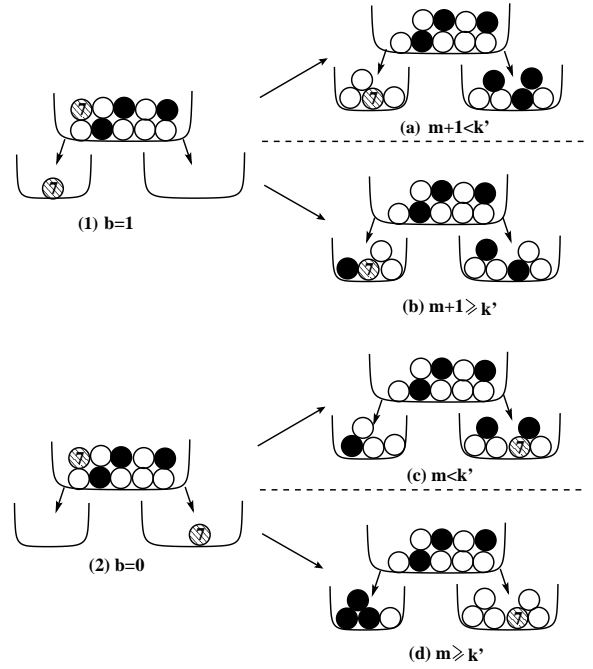


Figure 3: Illustration of TKD sampling.

the database size  $N$ , the rank of the item to be estimated  $k$ , the sample size  $n$ , the two Gamma distribution's parameters  $\alpha$  and  $\beta$ , and the number of Monte Carlo samples to obtain  $num$ , respectively.

Though simple, the naive algorithm is slow. The cost is  $O(N)$  per Monte Carlo iteration, where  $N$  is the database size. Since we are sampling for an extreme value in order to avoid scanning the entire data set (which is itself an  $O(N)$  operation), making use of an  $O(num \times N)$  Monte Carlo algorithm is unacceptable.

### 5.2 Practical Monte Carlo Sampling

Fortunately, we can do much better by simulating the two stages of the sampling process to produce a result that, statistically speaking, is indistinguishable from an actual execution of the naive method.

To reduce the cost of obtaining one Monte Carlo sample, we need to efficiently sample both  $f_{(k)}$  and  $\widehat{f}_{(k)}$ . It turns out to be easily possible to sample the order statistic  $f_{(k)}$  directly from its CDF (the details of how to do this are deferred to Section 5.4). Thus, the problem of sampling for the ratio  $\frac{\widehat{f}_{(k)}}{f_{(k)}}$  reduces to the problem of

sampling for  $\widehat{f}_{(k)}$  given a value of  $f_{(k)}$ .

To solve this reduced problem, we devise a Monte Carlo method called the *top-k dependent* (TKD) sampling technique that can efficiently produce  $\widehat{f}_{(k)}$  given  $f_{(k)}$ . The TKD method first determines whether or not the subsample includes  $f_{(k)}$  by means of a Bernoulli trial. Depending upon the result, the TKD method then figures out, in a randomized fashion, the composition of the  $k'$  largest items in the subsample; the  $(k')^{\text{th}}$  largest is then returned. Algorithm 3 formally describes the TKD method. The input parameters are identical to Algorithm 2, with the addition of the sampled  $f_{(k)}$ .

### 5.3 Example Use of the TKD Method

Since the TKD method has several different branches depending upon the results of the random samples obtained on lines (2), (4), and (13) of Algorithm 3, it is useful to illustrate the possible scenarios by means of an example.

---

**Algorithm 3** TKD Sampling

---

**Input:**  $N, k, n, \alpha, \beta$ , a single sample from  $f_{(k)}$

**Output:** a sample of  $\widehat{f_{(k)}}$ , corresponding to the input  $f_{(k)}$

```
1:  $k' = \lceil \frac{n}{N} \times k \rceil$ 
2: Let  $b \sim \text{Bernoulli}(x|\frac{n}{N})$  /*  $b$  is the result of a Bernoulli trial */
3: if  $b=1$  /*  $f_{(k)}$  is included in the subsample */ then
4:   Let  $m \sim \text{Hypergeometric}(x|N-1, k-1, n-1)$ 
5:   if  $m+1 < k'$  then
6:     Let  $S$  be  $n-m-1$  samples from a  $\text{Gamma}(x|\alpha, \beta)$  distribution, truncated above at  $f_{(k)}$ 
7:     Return the  $(k'-m-1)^{\text{th}}$  largest value in  $S$ 
8:   else
9:     Let  $S$  be  $m$  samples from a  $\text{Gamma}(x|\alpha, \beta)$  distribution, truncated below at  $f_{(k)}$ 
10:    Return the  $(k')^{\text{th}}$  largest value in  $\{f_{(k)}\} \cup S$ 
11:   end if
12: else
13:   Let  $m \sim \text{Hypergeometric}(x|N-1, k-1, n)$ 
14:   if  $m < k'$  then
15:     Let  $S$  be  $n-m$  samples from a  $\text{Gamma}(x|\alpha, \beta)$  distribution, truncated above at  $f_{(k)}$ 
16:     Return the  $(k'-m)^{\text{th}}$  largest value in  $S$ 
17:   else
18:     Let  $S$  be  $m$  samples from a  $\text{Gamma}(x|\alpha, \beta)$  distribution, truncated below at  $f_{(k)}$ 
19:     Return the  $(k')^{\text{th}}$  largest value in  $S$ 
20:   end if
21: end if
```

---

Using the previous section's notation, suppose that  $N = 10, k = 4, n = 4, k' = 2$  and  $f_{(4)} = 7$ . Figure 3 illustrates the four possible scenarios that TKD may encounter in processing these inputs. In Figure 3, each three-bucket group represents the result of either a Bernoulli or Hypergeometric trial. Balls represent samples, and the top bucket in each group contains the entire sampled query result set. The bottom-left bucket in each group contains the subsample, and the bottom-right bucket contains the remaining  $N-n$  samples. The black balls are the top- $(k-1)$  largest values in the query result set, and the shaded ball with a value label represents the  $k^{\text{th}}$  largest value over all.

The TKD method begins by first determining whether or not the  $k^{\text{th}}$  largest value appears in the subsample; this creates cases (1) and (2) in Figure 3, and corresponds to line (2) of Algorithm 3. In either case, it becomes necessary to determine the value of the  $(k')^{\text{th}}$  largest value in the subsample. In our example, in the case that the 7 appears in the subsample, we need to determine how many of the black (large-valued) balls also appear in the subsample. This is done via a call to  $\text{Hypergeometric}(x|9, 3, 3)$  on line (4) of Algorithm 3, which generates a Hypergeometric random variable with the specified distribution.

In Figure 3(a), this random trial determines that none of the black balls are retrieved by the subsample. Thus, the TKD method concludes that only one of the top- $k$  largest values are included in the subsample, which is less than  $k' = 2$ . Since we know that all the other items (the three white balls) on this side must be smaller than  $f_{(k)} = 7$ , in order to sample the second largest value from the subsample, the TKD method returns the largest of three samples from the truncated  $\text{Gamma}(x|\alpha, \beta)$  distribution, where  $x \in (0, f_{(k)} = 7)$ . A "truncated" distribution is simply a probability distribution with one of its tails chopped off.

However, this random trial could have determined that enough

black balls were contained in the subsample that the  $(k')^{\text{th}}$  largest value in the subsample is from the top  $k$  overall (line (8) of Algorithm 3). In Figure 3(b) the  $\text{Hypergeometric}(x|9, 3, 3)$  trial determines that one of the black balls is retrieved by the subsample. Thus, the subsample contains two of the top- $k$  values. The TKD method determines the black ball's value by drawing one additional sample from the truncated  $\text{Gamma}(x|\alpha, \beta)$  distribution, where  $x \in (f_{(k)} = 7, \infty)$ . From this set of two top- $k$  items, the second largest value is returned to the caller.

Two analogous situations occur if the Bernoulli trial has determined that the  $k^{\text{th}}$  largest overall does *not* appear in the subsample (lines (12)-(21) of Algorithm 3). In Figure 3(c), few enough black balls are included in the subsample that the  $(k')^{\text{th}}$  largest will come from the  $\text{Gamma}$  distribution truncated *above* at  $f_{(k)} = 7$ ; in Figure 3(d), enough black balls are included in the subsample that the  $(k')^{\text{th}}$  largest will come from one of the black balls, which are sampled from a  $\text{Gamma}$  distribution truncated *below* at  $f_{(k)} = 7$ .

## 5.4 Additional Technical Details

In this section, we address a few remaining technical issues regarding the Monte Carlo sampling process.

### 5.4.1 Sampling $f_{(k)}$

The process described above assumes that we have an efficient method to sample a value for  $f_{(k)}$  without having to generate the entire data set. To sample  $f_{(k)}$  efficiently, we can first obtain its CDF, which is defined by the following lemma:

**Lemma 1.** Given a query size  $N$ , a rank  $k$ , a shape parameter  $\alpha$ , and a scale parameter  $\frac{1}{\beta}$ , the CDF  $F_{f_{(k)}}$  for  $f_{(k)}$  is:

$$F_{f_{(k)}}(x) = \sum_{i=0}^{k-1} \binom{N}{i} [1 - F_{\text{Gamma}}(x)]^i [F_{\text{Gamma}}(x)]^{N-i}$$

where  $F_{\text{Gamma}}(x)$  denotes the CDF for the  $\text{Gamma}(x|\alpha, \beta)$  distribution.

**Proof:** Let  $Y$  be a random variable counting the number of values in the query result set that are greater than or equal to  $x$ . Thus,  $Y$  counts the size of the set  $\{f(d_i) \geq x\}$  for  $i \leq N$ . Since whether or not each  $f(d_i) \geq x$  is an independent Bernoulli trial, we see that  $Y$  follows the  $\text{Binomial}(N, 1 - F_{\text{Gamma}}(x))$  distribution. Then:

$$\begin{aligned} F_{f_{(k)}}(x) &= \text{Pr}[Y < k] \\ &= \sum_{i=0}^{k-1} \binom{N}{i} [1 - F_{\text{Gamma}}(x)]^i [F_{\text{Gamma}}(x)]^{N-i} \end{aligned}$$

■

Given the CDF of  $f_{(k)}$ , it is easy to sample  $f_{(k)}$  using the following two-step process:

1. Sample  $u$  from the  $\text{Uniform}(0, 1)$  distribution
2.  $f_{(k)} = F_{f_{(k)}}^{-1}(u)$

Step two can be implemented by solving for  $x$  in the equation  $u = F_{f_{(k)}}(x)$ . Since a CDF must be monotonically increasing, we can use binary search to obtain the solution. Note that the computation of  $F_{f_{(k)}}(x)$  requires  $O(k)$  time, which is fast since  $k \ll N$ .

### 5.4.2 Sampling From a Truncated Gamma

The TKD procedure needs to be able to sample an order statistic from a truncated  $\text{Gamma}$ , which ensures lines (6)~(7), (9)~(10), (15)~(16) and (18)~(19) of TKD algorithm use only  $O(k')$  time.

If a CDF for the truncated Gamma can be obtained, sampling an order statistic from the truncated Gamma can be implemented just as described by Lemma 1. Thus, it suffices to provide the CDFs for the required truncated Gamma distributions:

$$F(x) = \frac{\int_0^x \text{Gamma}(z|\alpha, \beta) dz}{\int_0^{f_{(k)}} \text{Gamma}(y|\alpha, \beta) dy}, \quad 0 < x < f_{(k)} \quad (7)$$

$$F(x) = \frac{\int_{f_{(k)}}^x \text{Gamma}(z|\alpha, \beta) dz}{\int_{f_{(k)}}^{\infty} \text{Gamma}(y|\alpha, \beta) dy}, \quad x > f_{(k)} \quad (8)$$

### 5.4.3 A Note Regarding the Scale Parameter

The reader may notice that we have omitted any mention as to how the inverse scale parameter  $\beta$  is obtained or dealt with by the Monte Carlo process. This may seem like a significant oversight, since  $\beta$  is not supplied. However, the “scale” is simply a multiplicative factor. Thus, since we are interested in the *ratio* of two values sampled from the same Gamma distribution, the scale is irrelevant. As a result, any of the Monte Carlo methods from this section can be implemented by simply choosing an arbitrary scale parameter larger than zero and using it consistently.

---

#### Algorithm 4 Approximating the Error Distribution

---

**Input:**  $p_j$  for  $j \in \{1, \dots, c\}$ ,  $MC_j$  for  $j \in \{1, \dots, c\}$ ,  $num$ ,  $x$   
**Output:**  $F_{ratio}(x)$

- 1: Sort  $MC = \bigcup_{j=1}^c MC_j$  in ascending order.
  - 2: For each entry in  $MC$ , if  $MC[i]$  came from shape pattern  $j$ , set  $MC[i].prob = p_j/num$ .
  - 3:  $i = 0$ ;  $tot = 0$ ;
  - 4: **while** (the current sample  $MC[i]$  is less than  $x$ ) **do**
  - 5:    $tot += MC[i].prob$
  - 6:    $i ++$
  - 7: **end while**
  - 8: Return  $tot$
- 

## 6. THE INFERENCE PHASE

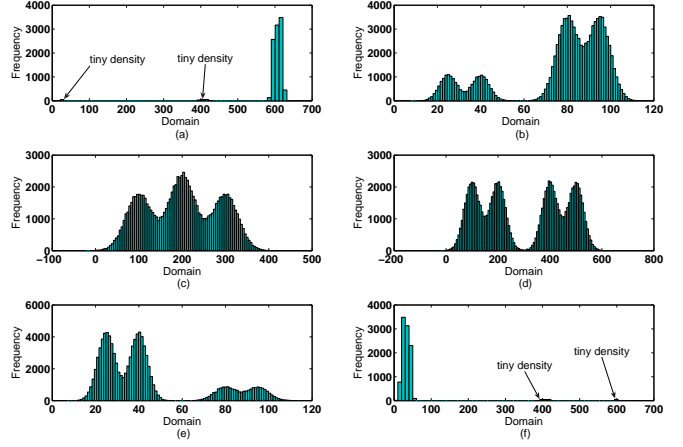
At this point, we have most of the tools necessary to complete the framework. Assume that we have completed the learning and characterization phases and have used a set of samples from a database to calculate  $\widehat{f_{(k)}}$ . We wish to characterize the distribution of  $\frac{f_{(k)}}{\widehat{f_{(k)}}$ .

Recall that the prior shape model consists of  $c$  weighted shape patterns. Given the same samples used to compute  $\widehat{f_{(k)}}$ , we can “update” those weights to incorporate the observed data using Bayes’ rule [13]. This is done by computing the posterior probability that we are sampling from the  $j^{th}$  shape pattern. In classic Bayesian fashion, the updated weight  $p_j$  is:

$$p_j = \frac{w_j p(\vec{q}|\alpha_j)}{\sum_{l=1}^c w_l p(\vec{q}|\alpha_l)}$$

Recall that  $\vec{q}$  denotes the aggregate triplet  $(M, S, n)$  corresponding to our database sample,  $w_j$  is the prior weight for shape pattern  $j$ , and  $p(\vec{q}|\alpha_j)$  is the PDF of shape pattern  $j$ .

Once the posterior weights are known, the final step in computing the distribution of the ratio  $\frac{f_{(k)}}{\widehat{f_{(k)}}$  is to combine the posterior weights with the Monte Carlo error distribution for each individual shape in order to compute a final, posterior distribution for the ratio. This is formalized in Algorithm 4, which details



**Figure 4: The histograms for the six synthetic query distributions considered in the first set of experiments; they are ordered from the easiest case to the hardest case.**

how to use this information to compute the total probability that  $\frac{f_{(k)}}{\widehat{f_{(k)}} < x$ . The arguments to the algorithm are, in order: the set of posterior weights  $p_1, p_2, \dots, p_c$ , all of the Monte Carlo samples  $MC_1, MC_2, \dots, MC_c$  (one set of samples for each shape pattern), the number of Monte Carlo samples for each shape pattern  $num$ , and finally the CDF input value  $x$ .

In this algorithm, all of the Monte-Carlo samples are first arranged in a sorted order from smallest ratio value to largest. The samples are then weighted according to the posterior weights. To calculate the probability that the ratio  $\frac{f_{(k)}}{\widehat{f_{(k)}}$  is less than an input  $x$ , we scan from the low end to the high end of the array  $MC$  and stop once we find the current Monte Carlo sample is greater than the input  $x$ . When we complete the scan, the sum of the probabilities processed will closely approximate the probability that  $\frac{f_{(k)}}{\widehat{f_{(k)}} < x$ . Given the ability to compute this probability, it is then trivial to associate bounds with the ratio  $\frac{f_{(k)}}{\widehat{f_{(k)}}$ .

## 7. EXPERIMENTS

This section details three sets of experiments. We first perform a set of experiments designed to test the accuracy and applicability of the Bayesian approach to extreme value estimation. Then we test application of the proposed Bayesian framework to two different, specific problems in the data management domain: approximate MAX (or top-k) aggregates and distance-based outlier detection.

### 7.1 Applicability of the Bayesian Approach

**Goals:** As discussed previously in the paper (particularly in Sections 3.2 and 4.1), there are some natural concerns regarding the application of the Bayesian approach to the problem of extreme value estimation. The experiments in this subsection are designed to directly test whether or not these concerns are legitimate. Specifically, we wish to answer two questions:

1. Since the shape model is derived from the Gamma distribution family, does our method actually work for query shapes that look nothing like a Gamma distribution?
2. Second, since the prior shape model is learned from the historical query workload, how will the Bayesian framework

Prior Weights	Sample size= 50				Sample size= 150				Sample size= 300			
	$k = 1$	$k = 5$	$k = 10$	$k = 20$	$k = 1$	$k = 5$	$k = 10$	$k = 20$	$k = 1$	$k = 5$	$k = 10$	$k = 20$
Uniform	0.84	0.86	0.81	0.85	0.91	0.92	0.91	0.93	0.95	0.95	0.95	0.96
Geometric(0.1)	0.84	0.86	0.82	0.85	0.90	0.92	0.91	0.92	0.95	0.95	0.94	0.96
Geometric(0.2)	0.78	0.80	0.77	0.80	0.87	0.88	0.88	0.90	0.93	0.93	0.93	0.95
Geometric(0.4)	0.86	0.89	0.85	0.88	0.92	0.93	0.93	0.94	0.95	0.96	0.96	0.97
Geometric(0.8)	0.88	0.89	0.86	0.89	0.93	0.94	0.94	0.95	0.96	0.96	0.96	0.97

Table 1: Coverage rates for 95% confidence bounds using sample size 50, 150, and 300, respectively.

function when the future query distribution has changed from the historical query distribution?

**Experimental Setup:** When evaluating these questions, the relevant metric is confidence bound coverage accuracy. That is, we wish to be able to ensure that no matter what the data look like and what the prior is, if the user specifies  $p\%$  confidence bounds, that the bounds we return are in fact  $p\%$  confidence bounds. To test the method’s robustness, we use six strangely-shaped, non-Gamma synthetic distributions to generate various query result sets. The generative distributions are illustrated in Figure 4. They are constructed to be multimodal, exhibit left and right skew with different degrees, and are discontinuous. Clearly, they bear little resemblance to the Gamma distribution. Using the discussion in Section 3.1, we order the shape patterns using the expected value of  $\frac{f_{(k)}}{\bar{f}_{(k)}}$ . The shape in Figure 4 (a) has the smallest expected value for this ratio, while the one in Figure 4 (f) has the largest.

Given these ordered shapes, we run a series of five tests. For each test, we begin by training our model using 500 randomly-generated queries, each having 1000 tuples sampled from one of the distributions illustrated in Figure 4. In the first test, the training queries are sampled uniformly. In the second, they are sampled according to a Geometric distribution with parameter 0.1, so that the first distribution in the ordered shape set is most likely to be sampled and the last one is least likely. In the third, fourth, and fifth, the Geometric parameters are 0.2, 0.4, and 0.8, respectively.

For each learned model, we then run 500 test queries, each of size 50000. The test queries are always sampled *uniformly* from the generative model. In this way, we test the case where the query generation is very different from the training distribution. For example, with a Geometric parameter of 0.8, we are very unlikely to see more than a few training queries from the last few distributions in Figure 4, though we will *test* quite often using those distributions (due to the uniform test generation). For each of the 500 test queries, we obtain 95% confidence bounds for the actual query answer using sample sizes of 50, 150, and 300, and also using four different  $k$  values. For each sample size, we compute the fraction of confidence intervals (coverage rate) that did, in fact, contain the actual query answer. These accuracies are given in Table 1.

**Discussion:** In general, the results show the high reliability of the Bayesian framework and clearly illustrate the robustness of the shape model. Using 300 samples, Table 1 shows almost perfect (95%) coverage in every case. There seems to be no dependence on the Gamma distribution (since the test data were clearly not Gamma-distributed) and very little (if any) dependence on the accuracy of the prior weights, since with sample size 300, Table 1 shows nearly perfect coverage no matter what the training distribution is (recall that the test distribution is always uniform).

One other interesting finding is that there *is* a dependence on sample size, since Table 1 shows coverage accuracy that is somewhat less than the expected 95% for extremely small samples. This is analogous to problems that occur when we have too few sam-

ples to obtain a good variance estimate using a classical estimation regime (see Section 3.2). However, we note that once a sample of size 300 has been obtained, the coverage is nearly perfect. We stress that 300 is a relatively tiny sample from a real-life database that may have billions of data points.

These findings are not surprising. Robustness to errors in the prior with an adequate sample size is a widely recognized merit of the Bayesian approach. As more and more samples are taken, the posterior distribution that we use to generate the bounds becomes less dependent on the prior distribution. It is generally acknowledged that in most circumstances, after a few hundred samples the prior carries little (if any) weight, and the sample is used almost exclusively to compute the result. Our results verify this supposition.

## 7.2 Approximate MAX (or Top-k) Aggregates

The most straightforward application of our Bayesian framework is using it to guess the largest value in a set. For example, one could easily use our Bayesian inference framework to facilitate an online answer to a top- $k$  selection query with an arbitrary selection predicate and aggregate function, along with accuracy guarantees. The records in the data set would be scanned in a randomized fashion, and at all times, the top  $k$  sampled records would be presented to the user. In order to give the user some idea of the quality of the answer set, the samples could be used to obtain confidence bounds on  $f_{(k)}$ . By comparing the  $k^{th}$  best record returned to the user with the bounds for  $f_{(k)}$ , the user may be given an idea of the quality of the answer set that has been obtained thus far.

### Application Details

The Bayesian inference framework developed in this paper could easily be used to provide the bounds on  $f_{(k)}$ . First, the previously observed query result sets, each represented by an aggregate triplet, would be used to train the prior shape model. During the training, all data points not accepted by a given training query are ignored when computing the query’s aggregate triplet. (that is, we ignore all  $f() = -\infty$  values). Also, all values not equal to  $-\infty$  are shifted so that their minimum value (the origin) is zero.

When it is time to evaluate a query, the samples from the new query result set are used. Again, we ignore all  $-\infty$  values. Given actual sample and database sizes  $n'$  and  $N'$ , for the purpose of our Bayesian framework we use  $n = |\{f(d) : f(d) \neq -\infty \wedge d \in DB\ sample\}|$  and  $N = \frac{N'}{n} \times n$ . That is, not only do we ignore  $-\infty$  values, but we “scale down” the size of the database to account for the expected number of  $-\infty$  values that cannot count towards  $f_{(k)}$ . To be consistent with the training, we also shift all of the sampled values so that the smallest value is also at the origin.

Next,  $\widehat{f}_{(k)}$  is computed from the sample. As described in the paper, the samples are also used to compute the posterior shape model, which in turn is used to compute the CDF  $F_{ratio}$  for the ratio  $\frac{f_{(k)}}{\bar{f}_{(k)}}$ . Given a confidence level  $p$ , bounds *low* and *high* are then chosen so as to minimize  $high - low$  subject to the constraint

Data set	Continuous/Feature	Size
Letter	16/17	20,000
CAHouse	7/9	20,640
El Nino	7/7	93,935
Cover Type	10/55	581,012
KDDCup99	34/41	4,898,430
Person90	12/13	5,000,000
Household90	7/11	5,523,522

**Table 2: Data description.** These data sets consist of both categorical and continuous features. Continuous/Feature denotes the number of continuous features over the number of total features.

that  $F_{ratio}(high) - F_{ratio}(low) = p$ . Finally,  $low \times \widehat{f}_{(k)}$  and  $high \times \widehat{f}_{(k)}$  are given as level- $p$  confidence bounds for the answer.

### Experimental Evaluation

**Goals:** When evaluating the utility of applying our Bayesian framework to this problem, there are two important questions to answer:

1. First, are the confidence bounds produced reliable for real-life data distributions, arbitrary selection predicates and aggregate functions, and various values of  $k$ ?
2. Second, how narrow are the bounds when a reasonably large sample has been obtained? That is, are they narrow enough to actually be useful?

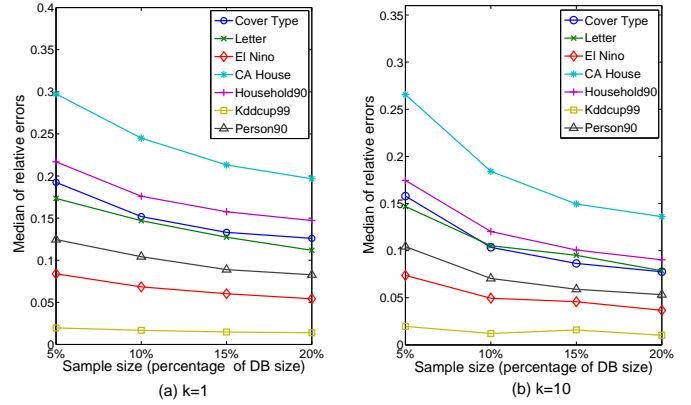
**Experimental Setup:** To test these questions, we selected seven real data sets, summarized in Table 2<sup>6</sup>. They are publicly available and span a range of problem domains with different characteristics. For each data set, a “query” is generated as follows. First, a tuple  $t$  is randomly selected. Then a selectivity  $s$  is randomly chosen from the range 5% to 20% and the  $s \times (data\ set\ size)$  nearest (Euclidean) neighbors of  $t$  are chosen as the actual query result set. The query’s aggregate function is defined as the weighted sum of three arbitrarily-chosen continuous attributes per query, where the weights are uniformly chosen from zero to one. The query answer is defined to be the  $k^{th}$  largest value of the aggregate function, as applied to tuples in the query result set.

For each data set, after training on 500 randomly selected queries using 10 shapes, 500 test queries are generated, and a 10% sample of the data set is used to provide 95% confidence bounds for the final answer to each query (we also experimented with using only 50 training queries; the results were nearly identical and so are omitted for brevity). Table 3 reports the observed coverage rates of the reported confidence bounds for various values of  $k$ .

In the second set of tests, we use  $k$  values of 1 and 10, respectively, and vary the sample size from 5% to 20% of each data set. We then compute the median relative confidence bound width at 95% (the relative confidence bound width is half the width of the bounds divided by the query answer). Figure 5 gives the results.

**Discussion:** In general, the confidence bounds provided show high reliability, which would seem to confirm the correctness of our framework and the appropriateness of a Gamma prior distribution for this problem, even for arbitrary, real-life data sets. There is *some* variation in coverage accuracy; for three of the seven data sets, the bounds were too conservative (showing coverage that was

<sup>6</sup>The first five data sets are from the UCI Machine Learning Repository. The last two data sets are from <http://usa.ipums.org>.



**Figure 5: Median relative confidence bound width of 500 test queries as a function of sample size.**

Data set	$k = 1$	$k = 5$	$k = 10$	$k = 20$
Letter	1.00	0.98	0.97	0.94
CAHouse	0.97	0.99	0.97	0.96
El Nino	1.00	1.00	0.99	0.99
Cover Type	1.00	1.00	0.99	0.99
KDDCup99	0.92	0.91	0.92	0.93
Person90	0.92	0.94	0.90	0.91
Household90	0.98	0.97	0.97	0.97

**Table 3: Coverage rates for 95% confidence bounds with various  $k$  values using a 10% sample.**

significantly higher than 95%), and for the KDDCup99 and Person90 data sets, the coverage accuracy was a bit lower than 95%. However, the confidence bounds overall were remarkably accurate given the difficulty of the problem. We feel that this is very strong evidence that the bounds generated will be safe and accurate given an arbitrary, real-life data set and query distribution.

The results shown in Figure 5 also demonstrate that depending on the data set in question, the bound width at 95% accuracy can be quite narrow, even for a 10% sampling fraction. For five of the seven data sets, a 10% sample provides for 95% bounds on the maximum value in the data set whose range is  $\pm 15\%$  of the actual query answer. Not surprisingly, this range generally shrinks as  $k$  grows, since a larger value of  $k$  means that we are trying to guess values that lie further from the extreme right tail of the distribution.

### 7.3 Distance-Based Outlier Detection

More generally, our framework is applicable to any problem where the goal is to find a few records in a set that are “close to” or “far away from” all of the other records in the set. In particular, this applies to distance-based outlier detection [11, 16, 2]. Given an arbitrary distance function  $dist$  (which may or may not be a metric distance), the goal is to pick the  $t$  ( $t \ll N$ ) database points whose distance to their  $k^{th}$  nearest neighbor ( $k^{th}$ -NN) is largest. The smallest distance in the result set is the *cutoff* distance.

#### Application Details

The state-of-the-art algorithm for outlier detection (due to Bay and Schwabacher [2]) is a nested loop algorithm (Algorithm 5). At all times, Bay and Schwabacher’s (Bay’s) algorithm maintains a result set. For each point in the data set, the algorithm checks to see if it can find more than  $k$  close-by points with respect to the current cutoff distance value. As soon as the algorithm can find enough

close points, the candidate outlier is pruned. Bay and Schwabacher have shown that if the points examined during the pruning step are considered in a randomized order, then excellent performance can be achieved.

---

**Algorithm 5** Bay and Schwabacher’s Nested Loop Algorithm

---

```

1: Initialize cutoff
2: Let Outliers = {}
3: for each point  $p \in D$  do
4:   Let countClose = 0
5:   for each point  $q \in D$ , in a random order do
6:     if  $\text{dist}(p, q) < \text{cutoff}$  then
7:       countClose ++
8:       if countClose  $\geq k$  then
9:         break the inner for loop and continue to process next
            $p$ 
10:      end if
11:    end if
12:  end for
13:  Add  $p$  to Outliers
14:  if  $|\text{Outliers}| > t$  then
15:    Remove the point from Outliers having the smallest
       $k^{\text{th}}$ -NN distance
16:    Set cutoff to be the smallest  $k^{\text{th}}$ -NN distance for any
      point in Outliers
17:  end if
18: end for

```

---

Our Bayesian framework can easily be used to reduce the number of distance computations required by this algorithm. To apply our framework, we view each database point as a query, and the set of distances from the point to each of the other points in the database is viewed as the point’s query result set. We randomly select a few points from the database as a training set and compute the distances from each training point to all of its neighbors. These queries are used to train a shape model. This model can then be used to speed Bay’s algorithm as follows:

1. First, we can carefully choose the order in which line (3) of Algorithm 5 considers the database points. If we can guess which points are outliers and consider them first, we will be sure that the cutoff value will be very large early on. This will increase the effectiveness of the algorithm’s pruning.

To choose such an advantageous ordering, we observe that the set of distances computed for training from each database point  $p$  to each of the training points is actually a sample of all of  $p$ ’s neighbor distances. This sample set can then be reused to compute  $\widehat{f_{(k)}}$  for  $p$ , as well as the distribution of the ratio  $\frac{\widehat{f_{(k)}}}{f_{(k)}}$  for  $p$ .<sup>7</sup> By computing the median value  $x$  for  $\frac{\widehat{f_{(k)}}}{f_{(k)}}$  (that is, the point  $x$  where  $F_{ratio}(x) = .5$ ), we can use  $x \times \widehat{f_{(k)}}$  as a guess for  $p$ ’s  $k^{\text{th}}$ -NN distance, and order the data set accordingly.

2. Second, we speed the algorithm by altering the loop of lines (5) to (12) by adding a second, probabilistic pruning condition after line (11). This additional pruning step is performed periodically (for example, after iterations 10, 20, 40,

---

<sup>7</sup>Note that in this application, because we are looking for nearest neighbors,  $f_{(k)}$  is used to denote the  $k^{\text{th}}$  smallest value in the set.

Data set	Speedup	Overlap	Error
Letter	2.61	25	0.02
CAHouse	3.22	28	0.00
El Nino	5.33	27	0.02
Cover Type	4.29	21	0.14
KDDCup99	3.92	24	0.02
Person90	5.28	24	0.07
Household90	4.29	28	0.00

**Table 4: Result of making use of the Bayesian framework within Bay and Schwabacher’s algorithm. For each data set, the table shows the speedup resulting from the application of the framework to the algorithm (Speedup), the size of the result set overlap between the “exact” version of the algorithm and the approximate one (Overlap), and the average relative error of the approximate version (Error).**

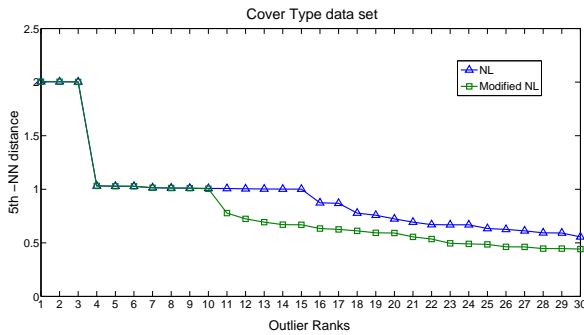
80, 160, and so on). Given all of the sampled neighbor distances that have been computed in line (6) for the particular point  $p$ , we use the Bayesian framework and the trained model to guess the distance from the point to its  $k^{\text{th}}$ -NN. If this guess is less than *cutoff*, then we can prune the point and still be reasonably sure that we have not pruned an outlier. In order to be “reasonably sure”, we should choose an upper bound on the  $k^{\text{th}}$ -NN distance that holds with high probability. In our implementation, we choose  $x$  so that  $\frac{\widehat{f_{(k)}}}{f_{(k)}}$  is less than  $x$  with 99% probability (that is, we choose  $x$  so that  $F_{ratio}(x) = 0.99$ ), and our guess as to the upper bound of  $k^{\text{th}}$ -NN distance is  $x \times \widehat{f_{(k)}}$ .

### Experimental Evaluation

**Goals:** We wish to experimentally test whether our Bayesian framework can be used to effectively speed Bay’s algorithm. There are two primary questions that we wish to answer:

1. First, what sort of speedup compared to Bay’s original algorithm can our framework help to provide?
2. Second, what exactly is the accuracy lost due to the probabilistic pruning that our modified algorithm provides?

**Experimental Setup:** We began our experiments by running Bay’s nested loop algorithm over the seven data sets from the previous subsection to obtain the top 30  $5^{\text{th}}$ -NN outliers. We record the total number of distance computations required as well as the actual answer set returned by Bay’s algorithm. Next, we run Bay’s algorithm augmented with our Bayesian framework as described above (making use of 80 training points and 10 shapes) and compute the speedup of the augmented algorithm with respect to the number of distance computations required. We also compute the overlap of the result set with Bay’s result set, as well as the average relative “error” of the augmented algorithm. For example, consider Figure 6, which shows the  $5^{\text{th}}$ -NN distances for the top 30 outliers discovered by both versions of Bay’s algorithm for the Cover Type data set (this is the data set where the augmented version of the algorithm returned the fewest “true” outliers). Let  $a_i$  be the  $5^{\text{th}}$ -NN distance for the  $i^{\text{th}}$  outlier returned by Bay’s algorithm, and let  $ab_i$  be the corresponding value for the augmented version of Bay’s algorithm. Then the average relative error is computed as  $\sum_{i=1}^{30} |ab_i - a_i| / \sum_{i=1}^{30} a_i$ . For each data set, the speedup, result set overlap size, and the average relative error are given in Table 4.



**Figure 6: Comparison of 5<sup>th</sup>-NN outlier distances for Bay and Schwabacher’s original nested loops algorithm, and the modified version of Bay and Schwabacher’s algorithm.**

**Discussion:** These experimental results show that by simply plugging our Bayesian framework into Bay’s outlier detection algorithm, one can generally obtain a factor of four improvement in running time. Furthermore, this improvement is obtained virtually for “free” and with almost no loss in result quality. In every experiment, the modified version of Bay’s algorithm returned more than 20 out of the 30 outliers that were returned by Bay’s original algorithm, but even that statistic tends to under-state the quality of the result. The actual difference in quality between the two result sets is always quite small; in five of the seven cases the average relative error is less than 2%. In the worst case (the Cover Type data set), the error is 14%, but close examination of Figure 6 shows that nearly all of this error is due to the loss of outliers 10 through 15, when it is unclear how much of a problem the loss of those few outliers might actually be.

## 8. RELATED WORK

Sampling and the use of other statistical methods have long been popular methods in databases and data management [8, 15, 10]. However, to date, sampling for extreme values has mostly been ignored as a data management research topic. The reason seems to be the difficulty of the problem. Unlike other aggregate functions such as COUNT, AVERAGE, MEDIAN, and so on, no universal limiting theorems such as the central limit theorem apply to extreme values.

Not surprisingly, statisticians have studied extreme values in detail, and many results are widely known. The essential reference book in this area is due to Leadbetter et al. [12]. At a high level, it is known that all parametric distributions fall into one of four classes with respect to the distribution of extreme values sampled from them. The most important class is those distributions whose largest values are distributed according to the Gumbel distribution. The normal distribution is a classic example of such a distribution. However, despite the large body of knowledge on this area in statistics, our work is quite different. Our goal was to develop a single method that is applicable to any, user-specified  $k$ , while most classical results from statistics (such as the Gumbel distribution) only apply to the max/min when  $k = 1$ . Another advantage of our method compared to straightforward application of classic statistical theory is that since we make use of a ratio estimator, our method is scale-agnostic. This means that our method can easily handle data sets with very large values and very small values in a single framework, without having to model each separately.

## 9. CONCLUSION

We have considered the problem of estimating the extreme values in a data set by looking at a small number of samples from it.

Because the relationship between the samples and the maximum (or minimum) value in a data set is so dependent upon the distributional properties of the data set in question, we have devised a unique, Bayesian framework for this problem that uses previously-observed queries to make a statistically rigorous guess as to the type of query that is currently under consideration. Significantly, we have given two examples of how this framework can be applied to various data management problems. The applications detailed are by no means exhaustive and it is far from clear that we have applied our framework in the best way possible for each problem. However, we believe our results conclusively show that this technique can be applied successfully to many other problem domains.

## 10. REFERENCES

- [1] L. Arge, O. Procopiuc, S. Ramaswamy, T. Suel, and J. S. Vitter. Scalable sweeping-based spatial join. In *VLDB*, pages 570–581, 1998.
- [2] S. D. Bay and M. Schwabacher. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In *KDD*, pages 29–38, 2003.
- [3] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty. *Nonlinear Programming: Theory and Algorithms*. Wiley, 1993.
- [4] J. Bilmes. A gentle tutorial on the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models. Technical Report University of Berkeley, ICSI-TR-97-021, 1997.
- [5] T. Brinkhoff, H.-P. Kriegel, and B. Seeger. Efficient processing of spatial joins using r-trees. In *SIGMOD*, pages 237–246, 1993.
- [6] D. Donjerkovic and R. Ramakrishnan. Probabilistic optimization of top n queries. In *VLDB*, pages 411–422, 1999.
- [7] P. J. Haas and J. M. Hellerstein. Ripple joins for online aggregation. In *SIGMOD Conference*, pages 287–298, 1999.
- [8] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. In *SIGMOD*, pages 171–182, 1997.
- [9] G. R. Hjaltason and H. Samet. Incremental distance join algorithms for spatial databases. In *SIGMOD*, pages 237–248, 1998.
- [10] W.-C. Hou and G. Özsoyoglu. Statistical estimators for aggregate relational algebra queries. *ACM Trans. Database Syst.*, 16(4):600–654, 1991.
- [11] E. M. Knorr, R. T. Ng, and V. Tucakov. Distance-based outliers: Algorithms and applications. *VLDB Journal*, 8(3-4):237–253, February 2000.
- [12] M. R. Leadbetter, G. Lindgren, and H. Rootzen. *Extremes and Related Properties of Random Sequences and Processes: Springer Series in Statistics*. Springer, 1983.
- [13] P. M. Lee. *Bayesian Statistics: An Introduction*. A Hodder Arnold Publication, 1997.
- [14] M.-L. Lo and C. V. Ravishankar. Spatial hash-joins. In *SIGMOD*, pages 247–258, 1996.
- [15] F. Olken. Random sampling from databases. Technical Report LBL-32883, 1993.
- [16] S. Ramaswamy, R. Rastogi, and K. Shim. Efficient algorithms for mining outliers from large data sets. In *SIGMOD*, pages 427–438, May 2000.
- [17] C. P. Robert and G. Casella. *Monte Carlo Statistic Methods*. Springer-Verlag, 2004.
- [18] H. Shin, B. Moon, and S. Lee. Adaptive multi-stage distance join processing. In *SIGMOD*, pages 343–354, 2000.