A One-Pass Space-Efficient Algorithm for Finding Quantiles

Rakesh Agrawal **Arun Swami***

IBM Almaden Research Center
650 Harry Road, San Jose, CA 95120

Abstract

We present an algorithm for finding the quantile values of a large unordered dataset with unknown distribution. The algorithm has the following features: i) it requires only one pass over the data; ii) it is space efficient — it uses a small bounded amount of memory independent of the number of values in the dataset; and iii) the true quantile is guaranteed to lie within the lower and upper bounds produced by the algorithm. Empirical evaluation using synthetic data with various distributions as well as real data show that the bounds obtained are quite tight. The algorithm has several applications in database systems, for example in database governors, query optimization, load balancing in multiprocessor database systems, and data mining.

1 Introduction

The p-quantile of an ordered sequence of data values is the smallest value below which p fraction of the total values in the sequence lie. Accurate estimation of the number of tuples satisfying a predicate is a prerequisite for a good query optimizer [11]. It was suggested in [9] that query optimizers should maintain information about attribute values distribution as p-quantiles, and estimation procedures were proposed using p-quantiles. Information about p-quantiles is also useful for balancing workload across multiple processors in a parallel database system [4].

*Current address: Silicon Graphics Computer Systems, 2011 N. Shoreline Blvd, Mountain View, CA 94043.

While probabilistic estimates for p-quantile values are adequate for some applications, others require *guaranteed bounds* on p-quantile values. For example, guaranteed bounds on p-quantile values are important in the operation of governors for relational database systems. Governors are used to provide feedback to users on response times for queries. A governor can either indicate the expected response time or provide some upper bound on the response time. For example, SmartMode from the IBI Inc. provides the first kind of feedback. If guaranteed bounds on response times are required, the governor must use guaranteed bounds on quantile values to estimate selectivities. The need for such predictive governors is a major user requirement.
As another example of an application of the guaranteed bounds on the p-quantile values, we cite the algorithm for mining association rules given in [1]. This algorithm concurrently synthesizes several functions of the form $f(A) \leq \tau^A$, while making a pass over a relation R, where A is some subset of attributes of R and τ^A is a cut-off threshold that depends on values of A in R. Computation of each of τ^A requires determination of a different p-quantile value, with guaranteed bounds.

1.1 Desiderata

The above cited applications yield the following desiderata for an algorithm for finding quantiles:

- It should not require prior knowledge of the data distribution.
- It should require only one pass over the data, since the data may be too large to fit in memory.
- It should be space efficient, since p-quantiles values of several attributes of the same relation may need to be obtained.
- The true p-quantile should be guaranteed to lie within the lower and upper bounds produced by the algorithm, and the bounds should be tight.

We present in this paper an algorithm that has *all* of the above features.

1.2 Related work

A straightforward method for finding p-quantiles is to sort the data and then make a pass over the sorted data to find the desired values. For large datasets, sorting requires multiple passes over the data and may incur large I/O costs besides the processing costs. Also, this procedure must be repeated for each attribute of the same relation for which p-quantiles are desired, making it computationally expensive. In [5], a technique that recursively uses a linear median finding algorithm [2] was proposed. It avoids external sorting and obtains accurate quantiles. However, this algorithm is not a one-pass algorithm and processes some data blocks multiple times.

The cost of finding quantiles can be reduced by using random sampling [3]. The idea is to take a random sample of the data, sort the sample values, and then use the sorted sample to estimate the p-quantile values. However, the p-quantile values produced by random sampling do not have guaranteed error bounds. Another algorithm for finding p-quantiles was proposed in [6]. This algorithm maintains a sorted stack of p-quantile estimated values and counts and updates those estimates as new data values are scanned using a piece-wise parabolic curve-fitting technique. Again, no guarantees can be given about the true p-quantile values using this technique.

Munro and Paterson [8] describe both single pass and multi-pass algorithms for determining quantiles (they call this the *selection* problem). Some of the algorithms are probabilistic and may fail to come up with correct bounds for the quantiles. The only deterministic single-pass algorithm they describe requires $O(N)$ storage, i.e., storage of the same order as the size of the data.

Another method for estimating quantiles works as follows. Let the range of values be partitioned into k intervals. These intervals are known as bins. A single pass is made over the data and the count of values in each bin is accumulated. The bins are scanned from lowest to highest values and the quantiles are determined based on the counts in the bins. In [10] one such method is analyzed. At first glance, this method appears to meet all our criteria. Only one pass is needed, the quantile is guaranteed to lie within a bin and modest computational and storage resources are required.

The problem is that unless the data distribution is known a priori, there is no simple way to determine what the boundaries of the bins should be. Determining the bin boundaries is almost as hard as determining the quantile values. In the absence of any knowledge, bins of equal width ranging from the minimum value to the maximum value can be used. A problem with this is that we may not know the minimum and maximum values. Sampling can be used to approximate these values but this requires more work. Even if the minimum and maximum values are known, problems can arise if values that are close together (fall into the same bin) are of high frequency. This scenario is common in real data where clusters of high frequency values tend to occur. Multiple frequent values in a single bin may result in highly inaccurate estimates for quantile values.

The algorithm we propose does not require a priori determination of bin boundaries. One way of viewing our algorithm is that it is a dynamic version of the algorithm in [10] in which the bin boundaries are determined on-the-fly and are continuously adjusted.

1.3 Formal Problem Statement

Quantile Problem. The p-quantile of an ordered sequence of data values is the smallest value in the sequence below which p fraction of the total values in the sequence lie. We can express the solution to the p-quantile problem in terms of another problem.

Let $X = \{x_i\}$ be a large unordered dataset, whose distribution is not known a priori. Denote by $\|X\|$ the cardinality of X. Let $\tau = p \times \|X\|$, $p \in (0, 1)$. Given a target count τ and assuming that $\|X\| \geq \tau$, we can define the following problem:

AL-LEQ Problem *(at least - less than or equal to)*. Find the smallest value v_τ in X such that $\|X_\tau\| \geq \tau$, where $X_\tau = \{x_i | x_i \in X \text{ and } x_i \leq v_\tau\}$.

The exact solution to the AL-LEQ problem solves the p-quantile problem. An *approximate* solution to the AL-LEQ problem would find a value v_τ in X satisfying all the conditions except that it may not be the smallest such value. Then, the approximate solution provides an upper bound on the p-quantile value. Now consider the following problem:

AM-LT Problem *(at most - less than)*. Find the largest value v_τ in X such that $\|X_\tau\| < \tau$, where $X_\tau = \{x_i | x_i \in X \text{ and } x_i < v_\tau\}$.

The approximate solution to the AM-LT problem provides a lower bound on the p-quantile value. Hence, if we can find approximate solutions to the AL-LEQ and the AM-LT problems, we have *guaranteed* bounds on the desired p-quantile.

If we solve the AM-LT and AL-LEQ problems in the same pass, we obtain the lower and upper bounds respectively for the p-quantile value of X. By solving these problems for different values of p in the same pass, we obtain different p-quantile values for X.

By symmetry, we can obtain lower and upper bounds for the p-quantile value by solving (approximately) the following problems:

AL-GEQ Problem *(at least - greater than or equal to)*. Find the largest value v_τ in X such that $\|X_\tau\| \geq \tau$, where $X_\tau = \{x_i | x_i \in X \text{ and } x_i \geq v_\tau\}$.

AM-GT Problem *(at most - greater than)*. Find the smallest value v_τ in X such that $\|X_\tau\| < \tau$, where $X_\tau = \{x_i | x_i \in X \text{ and } x_i > v_\tau\}$.

1.4 Outline of the paper

The rest of this paper is organized as follows. In Section 2, we describe a generic algorithm that can be customized to solve the problems described above. We prove that the algorithm is correct and show an example execution trace. In Section 3, we present an empirical evaluation of the algorithm using both synthetic and real data. We conclude with a summary in Section 4.

2 The Algorithm

We assume, for ease of exposition, that the input dataset X is a set of simple values. In general, X may be an n-ary relation and several functions (including accesser functions for attribute values) may have been defined, each of which maps the tuples of the relation to simple values. By maintaining a separate set of counters for each function, the proposed technique can be used for finding quantile values for all of these functions in a single pass. We first present a generic algorithm, and then show how it can be customized to solve the specific problems described in Section 1.3.

2.1 The Generic Algorithm

The algorithm given in Figure 1 uses a data structure H, which is an *ordered* list of k elements $e_1, e_2, \ldots, e_k$. Here k is the maximum number of elements in H, and we assume that $k \geq 2$. Each element e_i of H is a $(value, count)$ pair. We will use the notations $e_i.value$ and $e_i.count$ respectively to refer to the *value* and *count* fields of the list element e_i. The interpretation of these fields is that $e_i.count$ is approximately the count of sequence entries that fall between $e_i.value$ and $e_{i+1}.value$. The list H is initially empty. The operator θ is a generic comparison operator that will be made specific in Section 2.2.

In Figure 1, N_H^1 and N_H^2 need not be computed for every input value by iterating over all the elements in H. These two counts can easily be maintained incrementally. We have omitted these details in to avoid distracting the reader from the main ideas in the algorithm.

2.2 Customization of the Generic Algorithm

- AL-GEQ Problem:
  - Maintain H in increasing order of the *value* field of the elements.
  - The θ operator is the comparison operator $<$.
- AL-LEQ Problem:
  - Maintain H in decreasing order of the *value* field of the elements.
  - The θ operator is the comparison operator $>$.
- AM-GT Problem:
  - Set $\tau = \|X\| - \tau + 1$.
  - Solve the AL-LEQ Problem for the new value of τ.
- AM-LT Problem:
  - Set $\tau = \|X\| - \tau + 1$.
  - Solve the AL-GEQ Problem for the new value of τ.

H: An ordered list of maximum k (user-specified but ≥ 2) elements, initialized to be empty

```
forall values $x$ in $X$ do begin
    if there exists an element $e_i$ in $H$ suchthat $e_i.value = x$ then
        set $e_i.count = e_i.count + 1$
    else begin
        if number of elements in $H < k$ then
            insert ($x$,1) in $H$
        elsif $x \; \theta \; e_1.value$ then begin
            let $N_H^1 = \sum_{i=1}^k e_i.count$
            if $N_H^1 \geq \tau$ then
                discard $x$
            else begin
                let $e_l$ be the last element of $H$
                set $e_{l-1}.count = e_{l-1}.count + e_l.count$
                delete $e_l$
                insert ($x$,1) in $H$
            end
        end
        else begin
            find the last $e_i$ in $H$ such that $e_i.value \; \theta \; x$
            set $e_i.count = e_i.count + 1$
        end
    end

    let $N_H^2 = \sum_{i=2}^k e_i.count$
    if $N_H^2 \geq \tau$ then
        delete $e_1$
end

output $e_1.value$
```

AL-GEQ Problem: H increasing order, $\theta \equiv <$
AL-LEQ Problem: H decreasing order, $\theta \equiv >$
AM-GT Problem: $\tau = \|X\| - \tau + 1$, solve the AL-LEQ Problem
AM-LT Problem: $\tau = \|X\| - \tau + 1$, solve the AL-GEQ Problem

Figure 1: Generic algorithm and its customization

REMARKS:

- If the values are numeric, the AL-LEQ and AM-LT problems can be solved by solving the AL-GEQ and AM-GT problems for the negated values and vice versa. However, in general, X need not be numeric, so that the negation of the values in X is not defined but the values in X can be ordered and their p-quantiles determined.
- Multiple problems can be solved in a single pass by maintaining a separate H list for each problem.

2.3 Example

We show an example execution of the algorithm for the AL-GEQ problem in Figure 2. We are interested in finding the largest value v_τ in input X such that there are at least five values in X that are greater than or equal to v_τ (that is, target count $\tau = 5$). The list structure H is allowed to have a maximum of 3 elements (that is, $k = 3$). In this figure, the number before colon (`:`) is the input value and the list after colon shows the state of H. An element e of this list is represented as `V(C)` where $e.value = \texttt{V}$ and $e.count = \texttt{C}$. The algorithm returns 78 as the answer which happens to be the exact answer.

2.4 Complexity

The time complexity of the algorithm is $O(||X|| \cdot \log k)$ since any new value has to be inserted in the sorted list of size k. Each problem requires $2 \times k$ memory words. Our experiments indicate (see Section 3) that a small value for k is sufficient to estimate all p-quantiles with acceptable errors.

2.5 Correctness

We only discuss the correctness of the algorithm for the AL-GEQ problem. The correctness proofs for other problems are similar.

Lemma 1. Define for element e_j

$$N_H^j = \sum_{i=j}^{k} e_i.count$$

and

$$N_X^j = ||\{x_i \in X \text{ such that } x_i \geq e_j.value\}||$$

where both N_H^j and N_X^j are computed at the same point in time. Some or all of the x_i values in X have been read. Then

$$N_H^j \leq N_X^j$$

```
Target count = 5
Maximum number of elements in H, k = 3

Input Stream: 91 55 86 76 41 36 97 25 63 68 2 78 15 82 47

Execution:

91: 91(1)                // insert 91 (||H|| < k)
55: 55(1) 91(1)          // insert 55 (||H|| < k)
86: 55(1) 86(1) 91(1)    // insert 86 (||H|| < k)
76: 55(2) 86(1) 91(1)    // increment the count of 55
41: 55(2) 86(2)          // delete last, absorbing its count in previous
    41(1) 55(2) 86(2)    // insert 41
36: 41(1) 55(2) 86(2)    // discard 36 (36 < 41, total count >= target)
97: 41(1) 55(2) 86(3)    // increment the count of 86
    55(2) 86(3)          // delete first, count of remaining >= target
25: 25(1) 55(2) 86(3)    // insert 25 (||H|| < k)
    55(2) 86(3)          // delete first, count of remaining >= target
63: 55(2) 63(1) 86(3)    // insert 63 (||H|| < k)
68: 55(2) 63(2) 86(3)    // increment the count of 63
    63(2) 86(3)          // delete first, count of remaining >= target
 2:  2(1) 63(2) 86(3)    // insert  2 (||H|| < k)
    63(2) 86(3)          // delete first, count of remaining >= target
78: 63(2) 78(1) 86(3)    // insert 78 (||H|| < k)
15: 63(2) 78(1) 86(3)    // discard 15 (15 < 63, total count >= target)
82: 63(2) 78(2) 86(3)    // increment the count of 78
    78(2) 86(3)          // delete first, count of remaining >= target
47: 47(1) 78(2) 86(3)    // insert 47 (||H|| < k)
    78(2) 86(3)          // delete first, count of remaining >= target

Algorithm Returns: 78

Sorted Stream: 2 15 25 36 41 47 55 63 68 76 78 82 86 91 97

Correct Answer: 78
```

Figure 2: AL-GEQ Example

for all elements e_j in H at all times.
PROOF. At any time, if the algorithm discards an input value x, then $x < e_1.value$. In that case x does not contribute to a count of N_H^j or N_X^j for all j.
Otherwise, x contributes to a count of some element e_j in H. The element e_j is such that either $e_j.value = x$ or e_j is the last element in H such that $e_j.value < x$. In both the cases, for $1 \leq i \leq j$, the value N_H^i has increased by 1 and so has the value of N_X^i; for $i > j$, the value N_H^i of remains unchanged and so also the value of N_X^i.
When the algorithm deletes the last element e_l of H, $e_l.count$ is added to $e_{l-1}.count$. Thus, the value of N_H^i remains unchanged and equal to N_X^i for $i \in 1 \cdots l-1$.
To see that N_H^j and N_X^j are not always equal, consider the case when the algorithm deletes the first element e_1 of H, and the next value v falls between e_j and e_{j+1}. Then the value v is inserted in position $(j+1)$. Since the values between v and the old e_{j+1} have been "forgotten", we have that $N_H^{j+1} \leq N_X^{j+1}$. ∎

Lemma 2. After the processing of m input values,

$$N_H^1 = m, \text{ if } m \leq \tau$$

or

$$N_H^1 \geq \tau, \text{ if } m > \tau.$$

PROOF. No input value is discarded before τ values have been processed and they all contribute to a count in H. Once N_H^1 becomes equal to τ its value can be reduced only by the deletion of an element in H. There are two cases: the first element e_1 or the last element e_l of H is deleted. The first element is deleted only when the current value of $N_H^2 \geq \tau$. Then N_H^1 becomes equal to the current value of N_H^2 which we know is $\geq \tau$. When the last element e_l is deleted, $e_l.count$ is added to $e_{l-1}.count$. Hence the value of N_H^1 remains unchanged. ∎

Theorem. The algorithm in Figure 1 correctly solves the Al-GEQ problem.
PROOF. We have assumed that $||X|| \geq \tau$. After the set X has been processed, by Lemma 2 $N_H^1 \geq \tau$. By Lemma 1, $N_H^1 \leq N_X^1$. Thus, there are at least τ values in X that are greater than or equal to $e_1.value$. The algorithm correctly reports $e_1.value$ as the result. ∎

2.6 Additional Heuristics

It is possible to construct pathological cases for the proposed algorithm. The worst-case error for solving any of the four problems is $||X|| - ||H||$. For example, for the AL-GEQ problem, the worst case arises when the first $||H||$ values in the input data stream consists of the $||H|| - 1$ largest distinct values and the smallest value. In that case, the largest $||H|| - 1$ values will occupy $||H|| - 1$ positions of H and the smallest value will occupy the first position.

Several heuristics can avoid this worst case behavior. We present here three such heuristics. The basic idea behind these heuristics is to perturb H in such a way that correctness is not compromised and a position is opened up in H to avoid accumulating a large count at the first position. Here are the heuristics:

1. After every n input values, where n is an heuristically chosen parameter, do the following:

   $e_{l-1}.count = e_{l-1}.count + e_l.count$
   discard e_l

2. Suppose that the input value is such that it has incremented the count of e_1 in H. Now if

   $$e_1.count > n \times N_H^2$$

   where n is an heuristically chosen parameter, do the following:

   $e_{l-1}.count = e_{l-1}.count + e_l.count$
   discard e_l

3. Suppose that the input value is such that it has incremented the count of e_1 in H. Now if[1]

   $$e_1.count \geq n \times \tau$$

   then do the following:

   $e_{l-1}.count = e_{l-1}.count + e_l.count$
   discard e_l

Unfortunately, these heuristics do not always improve the accuracy. In fact, they can even degrade accuracy as the following example shows. Assume that $\tau = 5$ and $||H|| = 3$, and consider the following input stream for the AL-GEQ problem:

1 1 1 1 2 11 12 13 1 3 14 4 15

The algorithm in Figure 1 when applied to this data stream gives 11 as the answer, which is exactly correct. However, if Heuristic 3 with n set to 1 is applied to this data stream, we get 2 as the answer.

Fortunately, as the results below using both synthetic and real data show, the empirically observed accuracy of the proposed algorithm is very good. Hence, it does not seem worthwhile to incur the additional implementation complexity due to these heuristics. Our experiments also indicated that on average these heuristics did not improve the accuracy.

[1]We wish to thank Bruce Lindsay for suggesting a version of this heuristic for $n = 1$.

3 Empirical Evaluation

We conducted several experiments to empirically assess the behavior of our algorithm. We first show the results of experiments for X generated according to two distributions: the uniform distribution and the Zipf distribution [12]. For the Zipf distribution, we choose the Zipf parameter to be 0.86, which corresponds to the "80-20" distribution. We also experimented with other distributions by choosing different values for the Zipf parameter and found similar results.

The number of values in X ($||X||$) was one of $\{1$ million, 2 million, 5 million, 10 million$\}$. The number of distinct values was fixed at $||X||/10$. The values are positioned randomly in X. The p-quantile bounds were determined for the deciles, i.e., (10%, 20%, ..., 80%, 90%). Denote by N_e the number of tuples between the estimated bounds for each p-quantile. Denote by N_t the number of tuples between the true bounds for the corresponding p-quantile. Note that if the value of the p-quantile is duplicated m times then $N_t = m$. The relative error is computed as

$$(N_e - N_t)/||X|| \times 100$$

Note that our algorithm gives the bounds on p-quantile values. If a specific p-quantile value is desired, one can use either of the two bounds for this purpose. If the domain of input values is numeric, one can also take the average of the two bounds. The error we report is the sum of the two errors for the two bounds from the true p-quantile value. Hence, we are being conservative in our error measurement in the sense that we are reporting the sum of two errors. In practice, if we pick either of the bounds for the p-quantile value, we will incur only one of the errors.

A number of runs were carried out with the sequence of the data values being varied by changing the seed. For each run, the relative error described above is calculated. The error is estimated by averaging over the errors in the runs. We also calculated the 95% confidence interval. Sufficient runs were performed to ensure that the 95% confidence interval was less than 0.1%.

3.1 Varying the size of H

We varied the maximum size of H from 250 to 1000 in steps of 250, keeping $||X||$ fixed at 1 million. The results of this experiment are shown in Table 1. We observe that beyond a reasonable size for H, the error rates are quite flat. In the remaining experiments, we fixed the maximum size of H to be 750.

3.2 Varying the size of input

We now test the hypothesis that using a reasonable size of H (in our case 750) results in good error behavior for different stream sizes, i.e., for different values of $||X||$. In Table 2

Deciles	Uniform Distribution				Zipfian Distribution			
	H Size				H Size			
	250	500	750	1000	250	500	750	1000
10%	1.4	0.4	0.4	0.4	0.5	0.0	0.0	0.0
20%	0.5	0.4	0.3	0.2	0.0	0.0	0.0	0.0
30%	0.3	0.2	0.1	0.1	0.3	0.1	0.0	0.0
40%	2.3	0.6	0.4	0.1	0.5	0.4	0.3	0.1
50%	0.5	0.5	0.5	0.5	1.1	0.5	0.5	0.1
60%	0.6	0.5	0.5	0.5	0.9	0.5	0.4	0.4
70%	1.0	0.3	0.2	0.2	1.5	0.1	0.1	0.1
80%	0.0	0.0	0.0	0.0	0.7	0.2	0.2	0.1
90%	0.2	0.1	0.1	0.1	0.5	0.3	0.3	0.2

Table 1: Varying the size of H

Deciles	Uniform Distribution				Zipfian Distribution			
	$\|\|X\|\|$				$\|\|X\|\|$			
	1 million	2 million	5 million	10 million	1 million	2 million	5 million	10 million
10%	0.4	0.2	0.6	0.1	0.0	0.0	0.0	0.1
20%	0.4	0.4	0.1	0.2	0.0	0.5	0.3	0.3
30%	0.1	0.1	0.2	0.1	0.1	0.1	0.1	0.3
40%	0.6	0.1	0.4	0.1	0.4	0.2	0.2	0.4
50%	0.5	0.4	0.1	0.2	0.5	0.1	0.0	0.3
60%	0.5	0.6	0.4	0.2	0.4	0.3	0.1	0.3
70%	0.3	0.0	0.1	0.3	0.1	0.1	0.1	0.0
80%	0.0	0.6	0.1	0.5	0.2	0.1	0.4	0.1
90%	0.1	0.1	0.0	0.1	0.3	0.4	0.1	0.1

Table 2: Varying the size of input

we show how the errors measured change as $\|X\|$ varies from 1 million to 10 million.
We see that increasing the number of values in the stream X does not affect the error significantly even though the maximum size of H is kept fixed at 750. Assume that X is a set of four byte integers. Each count can be maintained in four bytes. Then, each entry in H is a eight byte entry, and we can determine all the deciles using about 0.6 megabyte of memory. This shows that using a small bounded amount of memory we can obtain accurate bounds on the p-quantiles. The algorithm becomes even more attractive as the number of values in the stream increases.

3.3 Comparison with Random Sampling

Next we compare the accuracy of our algorithm with random sampling, as random sampling is the technique most often used to estimate quantiles. Random sampling was given the same amount of memory for its sample as used by all the H structures needed for computing the decile values. The errors produced by the two algorithms are shown in Table 3 for $\|X\| = 1$ million and size of H set to 750.
We see that the accuracy of our algorithm is comparable to the random sampling. In addition, the true p-quantile values are guaranteed to lie within the bounds produced by our algorithm, something that random sampling and other probabilistic algorithms are unable to provide.

Deciles	Uniform Distribution		Zipfian Distribution	
	Our Alg	Sampling	Our Alg	Sampling
10%	0.4	0.1	0.0	0.1
20%	0.4	0.3	0.0	0.2
30%	0.1	0.5	0.1	0.4
40%	0.6	0.5	0.4	0.1
50%	0.5	0.5	0.5	0.1
60%	0.5	0.0	0.4	0.1
70%	0.3	0.1	0.1	0.3
80%	0.0	0.1	0.2	0.0
90%	0.1	0.2	0.3	0.0

Table 3: Comparison with Random Sampling: Synthetic Data

3.4 Reality Check

Finally, we assessed the accuracy behavior of our algorithm on data obtained from some customer databases, and compared it with random sampling. Data sets D1 through D5 had about 200,000 values and data set D6 had about 11.5 million values. The results are shown in Tables 4 and 5.

We see that our algorithm is somewhat more accurate than random sampling. We should note that the real data differs from the synthetic data we generated in that the real data has more duplicates. Also, the data distributions are not necessarily either uniform or Zipfian.

Deciles	Data Set D1		Data Set D2		Data Set D3	
	Our Alg	Sampling	Our Alg	Sampling	Our Alg	Sampling
10%	0.0	0.2	0.0	0.2	0.0	0.2
20%	0.0	0.3	0.1	0.2	0.5	0.0
30%	0.0	0.2	0.0	0.2	0.0	0.3
40%	0.0	0.3	0.0	0.3	0.1	0.2
50%	0.0	0.3	0.0	0.5	0.2	0.3
60%	0.0	0.2	0.0	0.3	0.2	0.3
70%	0.0	0.0	0.0	0.0	0.4	0.2
80%	0.0	0.3	0.0	0.4	0.0	0.2
90%	0.0	0.0	0.0	0.0	0.0	0.0

Table 4: Comparison with Random Sampling: Customer Data Sets D1, D2, D3

Deciles	Data Set D4		Data Set D5		Data Set D6	
	Our Alg	Sampling	Our Alg	Sampling	Our Alg	Sampling
10%	0.0	0.2	0.0	0.1	0.0	0.3
20%	0.1	0.2	0.0	0.1	0.0	0.3
30%	0.2	0.3	0.0	0.3	0.0	0.0
40%	0.0	0.2	0.0	0.2	0.0	0.0
50%	0.1	0.3	0.0	0.3	0.0	0.4
60%	0.2	0.3	0.0	0.0	0.0	0.1
70%	0.0	0.6	0.0	0.0	0.0	1.3
80%	0.1	0.0	0.0	0.1	0.0	0.0
90%	0.1	0.0	0.0	0.1	0.0	0.5

Table 5: Comparison with Random Sampling: Customer Data Sets D4, D5, D6

4 Summary

We presented an algorithm for estimating p-quantiles that has the following features:

- It does not require prior knowledge of the data distribution.
- It requires only one pass over data.
- The true p-quantile is guaranteed to lie within the lower and upper bounds produced by it.
- The bounds produced are quite accurate as shown by the errors observed in the experiments using both synthetic and real data.
- It is space efficient. Accurate results were obtained by using a small bounded amount of memory that is independent of the number of values in the dataset.

It was shown in [7] that in the case of queries involving multiple attributes, multi-dimensional equi-depth histograms are superior to equi-width histogram, and an algorithm based on multiple sorts was proposed for this purpose. Finding multi-dimensional equi-depth histograms is the same as finding p-quantiles in multi-dimensions. For future work, it would be interesting to explore how the algorithm proposed in this paper can be generalized to multi-dimensions. Finally, a probabilistic analysis of the proposed algorithm to characterize its average case behavior is a challenging open problem.

Acknowledgments. We thank Tomasz Imilienski for his participation in initial discussions. We also thank Peter Haas, Bruce Lindsay, Guy Lohman and Larry Stockmeyer for their comments.

References

[1] R. Agrawal, T. Imielinski, A. Swami: "Mining Associations between Sets of Items in Massive Databases," *ACM SIGMOD 93*, May 1993, 207–216.

[2] M. Blum *et. al*, "Time Bounds for Selection", *Journal of Computers and Systems*, **7**:4, 1972, 448–461.

[3] W. G. Cochran, *Sampling Techniques*, John Wiley and Sons, New York, NY, 3rd edition, 1977.

[4] D. J. DeWitt, J. F. Naughton, and D. A. Schneider, "Parallel Sorting on a Shared-Nothing Architecture using Probabilistic Splitting," *1st Int'l Conf. on Parallel and Distributed Information Systems*, Miami Beach, Florida, December 1991, 280–291.

[5] A. P. Gurajada and J. Srivastava, "Equidepth Partitioning of a Data Set based on Finding its Medians", Technical Report TR 90-24, Computer Science Dept., Univ. of Minnesota, 1990.

[6] R. Jain and I. Chlamtac, "The P^2 Algorithm for Dynamic Calculation of Quantiles and Histograms Without Storing Observations," *CACM*, Vol. 28, No. 10, Oct. 1985, 1076–1085.

[7] M. Muralikrishna and D. J. DeWitt, "Equi-Depth Histograms for Estimating Selectivity Factors for Multi-dimensional Queries," *ACM SIGMOD 88*, Chicago, Illinois, June 1988, 28–36.

[8] J. I. Munro and M. S. Paterson, "Selection and Sorting with Limited Storage," *Theoretical Computer Science*, Vol. 12, 1980, 315–323.

[9] G. Piatetsky-Shapiro, "Accurate Estimation of the Number of Tuples Satisfying a Condition", *ACM SIGMOD 84*, Boston, June 1984, 256–276.

[10] B. W. Schmeiser and S. J. Deutsch, "Quantile Estimation from Grouped Data: The Cell Midpoint," *Communications in Statistics: Simulation and Computation*, B6(3), 1977, 221–234.

[11] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lories, and T. G. Price, "Access Path Selection in a Relational Database Management System", *ACM SIGMOD 79*, June 1979.

[12] G. K. Zipf, *Human Behavior and the Principle of Least Effort*, Addison-Wesley, Reading, MA, 1949.