Range Query Estimation for Dirty Data Management System*

Yan Zhang, Long Yang, and Hongzhi Wang**

Department of Computer Science and Technology
Harbin Institute of Technology
zhangy@hit.edu.cn, {yanglonghit,whongzhi}@gmail.com

Abstract. In recent years, data quality issues have attracted wide attention. Data quality is mainly caused by dirty data. Currently, many methods for dirty data management have been proposed, and one of them is entity-based relational database in which one tuple represents an entity. The traditional query optimizations having the ability to estimate the cost of execution of a query plan have not been suitable for the new entity-based model. Then new query optimizations need to be developed. In this paper, we propose new query selectivity estimation based on histogram, and focus on solving the overestimation which traditional methods lead to. We prove our approaches are unbiased. The experimental results on both real and synthetic data sets show that our approaches can give good estimates with low error.

Keywords: query estimation, histogram, dirty data, data quality.

1 Introduction

Data quality has been addressed in different areas, such as statistics, management science, and computer science [1]. Dirty data is the main reason to cause data quality. Many surveys reveal dirty data exists in most database systems. The consequences of dirty data may be severe. Having uncertain, duplicate or inconsistent dirty data leads to ineffective marketing, operational inefficiencies, and poor business decisions. For example, it is reported [2] that dirty data in retail databases alone costs US consumers $2.5 billion a year. Therefore, several techniques have been developed to process dirty data to reduce the harm of dirty data.

Existing work on processing dirty data can be divided into two broad categories. The first category is data cleaning [3], which is to detect and remove errors and inconsistencies from data to improve data quality. However, data cleaning cannot clean the dirty data exhaustively and excessive data cleaning may lead to the loss of information. Besides this, existing data cleaning techniques are generally time-consuming. Therefore, some researchers propose algorithms in the other category, to

* This paper was partially supported by NGFR 973 grant 2012CB316200 and NSFC grant 61003046, 6111113089. Doctoral Fund of Ministry of Education of China (No. 20102302120054).

** Corresponding author.

H. Gao et al. (Eds.): WAIM 2012, LNCS 7418, pp. 152–164, 2012.

perform queries on dirty data directly and obtain query results with clean degree from the dirty data [4-6].

Several models for dirty data management without data cleaning have been proposed [7-9]. But most of these models only consider the uncertainty in values of the attributes and the quality degree of the data without the consideration of the entities in real world and their relationships. In this paper, we focus on entity-based relational database model in which one tuple represents an entity. This model can better reflect the real world entities and their relationships.

In applications, the different representations of the same real-word entities often lead to inconsistent data, uncertain data or duplicate data, especially when multiple data sources need to be integrated [10-11]. In the entity-based relational database, for the duplicate data referring to the same real-world entity, we combine these data, and for inconsistent data (or uncertain data), we endow each of them a value (we call it as quality degree) which reflects its quality. *Example 1* shows this process.

Table 1. A Dirty Data Fragment

ID	*Name*	*City*	*Zipcode*	*Phn*	*Reprsnt*
1	Wal-Mart	Beijing	90015	80103389	Sham
2	Carrefour	Harbin	20016	80374832	Morgan
3	Wal-Mart	BJ	90015	010-80103389	Sham
4	Walmart	Harbin	20040	70937485	Sham
5	Carrefour	Beijing	90015	83950321	Morgan
6	Mal-Mart	Beijing	90015	80103389	Sham

Example 1: *Consider a fragment of the dirty data shown in Table 1. We can easily identify that tuples 1, 3 and 6 refer to the same entity in the real world even though their representations are different. By preforming entity resolution and combining these three tuples, we can get one entity tuple. In this process, we don't remove any data, which implies that the value of one attribute in a tuple may be uncertain, and it may contain multiple values. We endow possible each attribute value with a quality degree in accordance with their proportion, as shown in Table 2. In tuples 1, 3 and 6, the value "Wal-Mart" appears twice, so the quality degree is 2/3 ≈ 0.67. Similarly, other quality degrees can be given. Then we get an entity tuple as shown in Table 2.*

Table 2. An Entity Tuple

ID	*Name*	*City*	*Zipcode*	*Phn*	*Reprsnt*
1	(Wal-Mart, 0.67), (Mal-Mart, 0.33)	(Beijing, 0.67), (BJ, 0.33)	(90015, 1.0)	(80103389, 0.67), (010-80103389, 0.33)	(Sham, 1.0)

As *Example1* shows, the entity-based relational database ingeniously processes the dirty data by entity resolution [12-13] and quality degree. In the implementation of this model, query optimization techniques are in demand. As the base of the query optimization, the estimation technique computing the size of the results of an operator is crucial. Even though over the past few decades, there has been a lot of work on query estimation for traditional relational database management systems. Most

approaches for query estimation are based on histogram [14], which records data distributions. However, the traditional histograms are not suitable for entity-based relational database, and often lead to overestimation, especially for range queries. One reason is that query processing on the entity-based relational database need to consider the effect of the quality degrees of values, but the traditional histograms are only concerned about the attribute value without the consideration of the quality degree. The other reason is that traditional approaches based on histogram often lead to overestimation, especially for range queries. Because one attribute of a tuple may contain multiple values in the entity-based relational database, if all values are partitioned into different buckets, this tuple is counted for multiple times. Thus, the overestimation occurs.

Therefore, the traditional query estimation approaches based on histogram cannot be applied to our problem. Unfortunately, there is no work for the query estimation of the entity-based relational database. New query estimation approaches are in demand.

Our Contributions: In this paper, we propose new range query estimation methods suitable for entity-based relational database. As we know, this is the first paper considering such problem. These algorithms are demonstrated in details and the complexity of these algorithms is analyzed. We theoretically prove our algorithms are unbiased. Last, we experimentally validate the effectiveness of our algorithms and show that our methods are accurate.

The rest of this paper is organized as follows. Section 2 introduces entity-based relational database model and some related conceptions. Section 3 presents our range query estimation methods. We show our experimental results in Section 4. We conclude our paper and discuss the future work in Section 5.

2 Preliminaries

2.1 Entity-Based Relational Database Model

We firstly define the *Uncertain Attribute Value* in Definition 1. An uncertain attribute value not only contains possible values, but also contains the corresponding quality degrees. Then we give the definition of *Entity* in Definition 2. Entity is the basic unit of storage in the entity-based relational database system, containing a set of uncertain attribute values.

Definition 1 (Uncertain Attribute Value): *An uncertain attribute value is a set of pair* $A = \{(v, p) | v$ *is possible value of the attribute and p is the quality degree of the value v}.*

Definition 2 (Entity): *An entity is a pair* $E = (K, A)$*, where* A *is a set of uncertain attribute values and* K *is a set of keys that is to identify the entity uniformly (e.g., entity-ID).*

Table 2 can help to understand these two definitions. Since we introduce the quality degree dimension, we need to define a new conception to reflect whether a tuple satisfies a query, and we call this conception *Similarity*.

Definition 3 (Similarity): *For an uncertain values V in attribute a and an atom constraint C in form of a @ v where @ is a predicate symbol (e.g. >, < …) and v is a constraint, the similarity between them is defined as follows:*

$$Sim(V@C) = \sum_{(v_i,p_i)\in V} sim(v_i@v)p_i \ , \tag{1}$$

$$sim(v_i@\ v) = \begin{cases} 1 & where\ v_i@v \\ 0 & otherwise \end{cases} . \tag{2}$$

For a selection query with a constraint $a < v$ (for the convenience, we use this form $a < v$ to represent a selection query in this paper), we consider that one tuple satisfies query with a similarity S, which can be calculated by Equation (1) and (2). We use an example to illustrate it.

With the support of the conceptions, some query operators are defined.

2.2 Operators

In this paper, we focus on the estimation of selection operation. Each query result satisfies the query with a similarity, since results with a low similarity are generally less interesting than higher similarity answers, we consider those results with a similarity less than a threshold τ (this parameter can be provided by user or the system sets a default value) as unsatisfied for a query. Therefore, the results of queries should be those answers that have a similarity exceeding a threshold τ. So a query given by $a <_\tau x$ can be defined as an operator as follows:

$$Sim(a < x) > \tau \Leftrightarrow \sum_{(v_i,p_i)\in V} sim(v_i < x)p_i > \tau \ . \tag{3}$$

The goal of this paper is to propose new query estimation techniques for the entity-based relational database. In next section, we will give our approaches in details.

3 Range Query Estimation

In this section, we describe our estimation methods in details. First，we give a preliminary query estimation method in Section 3.1, and this method can well estimate unbounded range query (e.g., $a >_\tau x$ or $a <_\tau x$) result size, but for general range queries (e.g., $x_1 < a < x_2$), it often leads to underestimation. Then, a more accurate range query estimation method is proposed in Section 3.2, and it can well solve the underestimation problem which the former method encounters.

3.1 Preliminary Range Query Estimation Method

As discussed in Section 1, existing query estimation methods are not suitable for range queries on entity-based relational database management system for two reasons that with the quality degree, the existing methods often lead to the overestimation. To solve these problems, we consider an unbounded range query Q by $a <_\tau x$ firstly, where a is an uncertain attribute value and τ is the similarity threshold. This query returns all tuples satisfying $Sim(a < x) > \tau$, which means that a satisfies the following relationship:

$$\sum_{(v_i,p_i)\in V} sim(v_i < x)p_i > \tau \ . \tag{4}$$

If all possible values of an uncertain value are sorted in database system, the relationship (3) is equivalent to calculate the cumulative distribution function $F_a(x)$, where $F_a(x) = \sum_{v_i<x} p_i$, and return the values satisfying $F_a(x) > \tau$.

Fig. 1 shows an example of the cumulative distribution functions (CDF) of several tuples on attribute A, whose corresponding values are shown in Table 3. In the figure, each stacked line represents one tuple. The meaning of every stacked line is like the cumulative distribution function of every uncertain value. For example, the point *P* on the stacked line represents that the value of attribute A of tuple 3 is smaller than 35 with similarity 0.6. Therefore, with such a figure containing all tuples, for a given query Q ($a <_\tau x_0$), the total number of tuples which satisfy query Q can be estimated directly. It is the number of stacked lines crossing the line segment l given by $x = x_0, \tau < y \leq 1$.

Table 3. A Data Fragment

ID	*A*	*B*	·····
1	((10, 0.1), (35, 0.3), (65, 0.5), (80, 0.1))	·····	·····
2	((20, 0.3), (50, 0.5), (80, 0.2))	·····	·····
3	((15, 0.6), (60, 0.2), (70, 0.2))	·····	·····

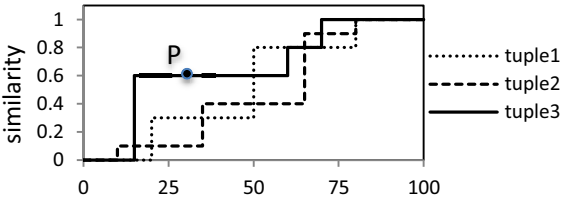

Fig. 1. Example for showing the histogram structure

Histogram Structure

Based on the above discussion, we define a basic two-dimensional histogram. The range of input values is partitioned into $n * m$ buckets where n and m are the lengths of each dimension. A histogram bucket $H(i,j)$ covers the area given by $(i * \delta_x,\ j * \delta_s,\ (i+1) * \delta_x,\ (j+1) * \delta_s)$, where $\delta_x$and δ_s are the widths of histogram bucket along *x* and *y* axis. Each bucket has a value, which stores the height of this bucket that records the number of tuples whose stacked lines intersect this bucket.

Obviously, the errors of the estimations using this histogram are associated to the number of the stacked line inflection points in buckets and do not exceed them. If there is no inflection points in bucket H_i, the estimation for queries which are located in H_i must be accurate. Hence in order to make the estimation more accurate, we need to ensure that the number of the inflection points in each bucket H_i is small enough.

In our approach, the histogram is firstly partitioned into p equal-width buckets, and we set the number of the inflection points in each bucket should not exceed ε

($\varepsilon = M/p$, where M is the total number of inflection points, which equals the number of all possible attribute values). When a bucket contains more than ε inflection points, this bucket is partitioned into q equal-width buckets (generally, $q \ll p$, and q can be considered as a constant) and we set each new bucket containing ε/q inflection points. In the next process, for the buckets which do not meet the requisition, they are partitioned until that all buckets contain less than ε inflection points.

We now present the histogram construction algorithm. To facilitate the description of algorithm, we firstly summarize the main notations that will be used in our paper in Table 4. With these notations, *Algorithm 1* illustrates the detailed steps of histogram construction. Note that, we assume all possible values of an uncertain value are increasing in the database system. For each uncertain value, it is supposed that all possible values are $v_0, v_1 \ldots v_{m-1}$ and the corresponding quality degrees are $p_0, p_1 \ldots p_{m-1}$, the interval $[v_{min}, v_{max}]$ can be divided into $m+1$ intervals: $[v_{min}, v_0)$, $[v_0, v_1) \ldots [v_{m-1}, v_{max}]$. In each interval, we need to record value a in correct histogram buckets. For example, in $[v_{min}, v_0)$, value a should be recorded in buckets $H([v_{min}, v_0), 0)$, which represents $Sim(a < x)$ is 0, where $x \in [v_{min}, v_0)$. Similarly, value a should also be recorded in buckets: $H([v_0, v_1), p_0), H([v_1, v_2), p_0 + p_1) \ldots H([v_{m-1}, v_{max}], 1)$. Meanwhile, the number of inflection points is stored in each bucket, and when the size of some bucket exceeds ε, it is partitioned into q equal-width buckets and the histogram is adjusted. *Algorithm 1* is the pseudo-code of this process, where symbol P_i represents the number of inflection points in bucket H_i.

Table 4. Main Notations

Notation	Meaning
τ	Similarity threshold
p, q	Initial granularity of partition and granularity of repartition
ε	Threshold of the number of inflection points in one bucket
l_i, r_i	Left boundary and right boundary of bucket H_i along *x* axis
δ_s, s	Width of buckets along *y* axis, where $s = \lceil 1/\delta_s \rceil$
a_i	Uncertain values of attribute *A* ($0 \le i < N$)
v_i, p_i	Possible values and quality degrees of an uncertain attribute value *a*
v_{min}, v_{max}	Minimum and maximum among all possible values of an attribute *A*
$sim(x_1, x_2)$	Similarity of $x_1 < a < x_2$ where *a* is an uncertain value
C	The number of tuples satisfying query Q

Algorithm 1

```
1  Initialize H ← ∅
2  for each uncertain value a do
3      for each possible value v_k of an uncertain value do
4          for all buckets meeting v_{k-1} < l_i < v_k do
5              H(i, ⌊F_a(l_i)/δ_s⌋)++
6          P_i++
7          if P_i > ε then
8              partition and adjust this bucket
9      for all buckets meeting r_i > max (v_k) do
10         H(i, ⌈1/δ_s⌉)++
```

Theorem 1: *The time complexity of Algorithm 1 is* $O(p(N+s))$ *and the space complexity is* $O(ps)$.

Proof: This algorithm scans each tuple once and records each tuple in *m* appropriate histogram buckets, where *m* is the length of the histogram in *x* dimension. In the worst case, partition occurs per $\varepsilon(q-1)/q$ tuples, and partition times does not exceed $pq/(q-1)$ (i.e., $M/(\varepsilon(q-1)/q)$). Each partition adds $q-1$ buckets and adjusts s buckets along *y* axis, so $m < (q-1)pq/(q-1)+p = p(q+1)$. Thus the time complexity is $O(p(q+1)N)+O(qspq/(q-1))$ and the space complexity is $O(p(q+1)s)$. Thus the time complexity is $O(p(N+s))$ and the space complexity is $O(ps)$, because q can be considered as a constant. □

Query Estimation Method

With the histogram structure, we can easily estimate query result size. Given a query $a <_\tau x_0$, query result size is estimated as the sum of the heights of the buckets where x_0 is located in and meet the similarity threshold. *Algorithm 2* shows this algorithm in details.

However, this algorithm is only applicable for the unbounded range queries in form of $a <_\tau x_0$. For anther unbounded range queries in the form of $a >_\tau x_0$, we need to perform an equivalent transformation to make *Algorithm 2* suitable for such form.

$$a >_\tau x_0 \Leftrightarrow Sim\,(a > x_0) > \tau \Leftrightarrow Sim\,(a < x_0) < 1-\tau \quad . \tag{4}$$

Such that *Algorithm 2* can also be used to estimate queries in the form as $a >_\tau x_0$, with a modification of the loop range in *line 4*, and the loop range should be modified to $[0, \lfloor(1-\tau)/\delta_s\rfloor]$. Theorem 2 proves this estimation method is unbiased when p tends to infinity.

Algorithm 2

```
1  if x_0 < v_min then return 0
2  if x_0 > v_max then return N
3  let C = 0 and find H_i meeting l_i < x_0 ≤ r_i
4  for j from ⌊τ/δ_s⌋ to ⌈1/δ_s⌉ do
5      C = C + H(i, j)
6  return C
```

Theorem 2: *The estimation method in Algorithm 2 is unbiased when* p *tends to infinity.*

Proof: To facilitate the proof, we assume $q = 2$, for other cases, the proof process is similar. As proved in Theorem 1, partition times does not exceed $2p$ (i.e., $pq/(q-1)$), and each partition adds 1 (i.e., $q-1$) buckets. Given a query $a <_\tau x_0$, we make the following assumptions. First, x_0 falls each bucket with equal probability. Second, n times partitions occur. Last, m buckets contain more than ε inflection points where $m \le n$. With these assumptions, the total number of buckets along *x* axis is $p+n$. We have known the estimation error does not exceed the number of inflection points in bucket which x_0 is located in. Hence the expectation of estimation error is:

$$E(e) < \frac{p+n-m}{p+n}\varepsilon + \frac{m}{p+n}E'(e) = \frac{p+n-m}{p+n}\varepsilon + \frac{m}{p+n}\left(\varepsilon + \frac{M-m\varepsilon}{m}\right)$$
$$= \varepsilon + \frac{M-m\varepsilon}{p+n} < \varepsilon + \frac{M}{p} = 2\frac{M}{p}.$$

Therefore, when p tends to infinity, the expectation of estimation error tends to 0, and this approach is unbiased. □

We have discussed the unbounded range queries. Consider the general range query Q ($x_1 < a < x_2$), and that is $Sim(x_1 < a < x_2) > \tau$. The unbounded range queries can be considered as a special case of the general range query. To estimate the general range query result size, with the application of the techniques in this section, a naïve method is proposed. Firstly, the numbers of tuples that satisfy query Q1 ($a <_\tau x_1$) and query Q2 ($a <_\tau x_2$) are estimated by *Algorithm 2*, and they are denoted by $C1$ and $C2$ respectively. We can use $C2 - C1$ as the estimation of query Q ($x_1 < a < x_2$). Clearly, if we do not consider the threshold, this method is correct. However, it often leads to underestimation with the consideration of the effect of the similarity threshold on query result sizes, and it is related to the width of query range and the threshold. We show the effect by experiments in Section 4. With the shortcoming of this naïve method, we propose more accurate query range estimation method in next section.

3.2 Accurate Range Query Estimation

In this section, we present an accurate range query estimation algorithm, and it can solve the underestimation problem discussed in Section 3.1. In order to adapt to general range queries, we add another dimension to the histogram proposed in Section 3.1. The meanings of two original dimensions do not change (the *x* axis and *y* axis respectively represent the end point of the query and the similarity), and the new additional dimension (*z* axis) represents the beginning of the query. Therefore, given a general range query $Q(x_1 < a < x_2)$, we can estimate the size of query result set by counting the number of stacked lines crossing the line segment l given by $x = x_2$, $\tau < y \leq 1$ and $z = x_1$, similar to Fig. 1. That is equivalent to executing a query Q' ($a <_\tau x_2$) on the plane, where $z = x_1$.

We call such new histogram as improved histogram. In this histogram, every plane on *z* axis is a basic histogram proposed in Section 3.1, corresponding to the constraint $z \leq x < v_{max}$ (clearly, it is not necessary to store the whole range). The width of a bucket on *z* axis is controlled by an input parameter δ_z(in general, δ_z can be equal to $(v_{max} - v_{min})/p$). The detailed algorithms for constructing this improved histogram and estimating the result size of a general range query are respectively presented in *Algorithm 3* and *Algorithm 4*. Compared with *Algorithm 1*, *Algorithm 3* only adds another layer of loops on *z* axis, but this improved histogram structure can give more accurate estimation than the basic histogram of Section 3.1. Theorem 3 gives the time and space complexity of the construction algorithm, and Theorem 4 proves this estimation algorithm using this improved histogram is also unbiased when p tends to infinity.

Algorithm 3

```
1  Initialize H ← ∅
2  for each uncertain value a do
3      for each possible value v_n of an uncertain value do
4          for k from 0 to ⌊(v_k − v_min)/δ_z⌋ do
5              for all buckets meeting v_{n−1} < l_i < v_n do
6                  H(k, i, sim(k ∗ δ_z, l_i)/δ_s)++
7              P_{k,i}++
8              if P_{k,i} > ε then
9                  partition and adjust this bucket
10     for k from 0 to ⌊(v_k − v_min)/δ_z⌋do
11         for all buckets meeting r_i > max (v_k) do
12             H(k, i, sim(k ∗ δ_z, l_i)/δ_s)++
```

Theorem 3: *The time complexity of Algorithm 3 is* $O(p^2(N+s))$ *and the space complexity is* $O(p^2 s)$ *with the assumption:* $\delta_z = (v_{max} - v_{min})/p$.

Proof: Compared with *Algorithm 1*, this algorithm only adds another dimension, and the length of this dimension is p. Therefore, similarly the analysis of the complexity of *Algorithm 1*, the time complexity of *Algorithm* 3 is $O(p^2(N+s))$ and the space complexity is $O(p^2 s)$. □

Theorem 4: *The estimation method in Algorithm 4 is unbiased when* p *tends to infinity.*

Proof: Compared with the basic histogram of Section 3.1, this improved histogram with more detailed information can get a more accurate estimation for general queries. Therefore, with the conclusion of Theorem 2, the estimation method using this improved histogram is also unbiased when p tends to infinity. □

Algorithm 4

```
1  if x_1 < v_min then let x_1 = v_min
2  if x_2 > v_max then let x_2 = v_max
3  let C = 0; k = ⌊(x_1 − v_min)/δ_z⌋ and find H_{k,i} meeting l_i < x_2 ≤ r_i
4  for j from ⌊τ/δ_s⌋ to ⌈1/δ_s⌉ do
5      C = C + H(k, i, j)
6  return C
```

4 Experimental Evaluation

In this section, we study the performance of our proposed algorithms experimentally. Our experiments are conducted on a 2.93 GHz Inter(R) Core(TM)2 Duo CPU with 2 GB main memory.

4.1 Data Sets

The data sets used for estimate query result size can be categorized into two main parts of synthetic data sets and real-world data sets. Table 5 summarizes some information about these data sets.

Synthetic Data Sets: We generate the synthetic data sets and each tuple has a Tuple ID, along with an uncertain value. The number of the possible values of an uncertain value is uniformly distributed between 1 and 5. The quality degree of each possible value is randomly generated from 0.01 to 1 and these quality degrees sum up to 1 for an uncertain value. To evaluate the robust of our approaches, we consider three synthetic data sets with different distributions: uniform distribution, normal distributions and zipfian distribution.

Real-World Data Sets: One of the most important applications of the histograms is for those cases in which the distribution of the data is unknown or cannot be simply modeled. Therefore, in order to validate our approaches over such kind of data, we consider the real-world data sets: eCommerce data. We respectively collect book information about *Computers & Technology* from eBay (http://www.ebay.com) and Amazon (http://www.amazon.com). After the processes for original data, we get the real data set with 10053 entities. In this data set, each tuple represents a book which contains four uncertain values: *title*, *author*, *press* and *price*. We perform our experiments by building the histograms on attribute *price*.

Table 5. Data sets used for the experimental results

Name	Distribution	Size	Parameter
Uniform	Uniform	$1m$	$min0, max1k$
Normal	Normal	$1m$	$\mu 500, \sigma 100$
Zipf	Zipfian	$1m$	$\alpha 1.0$
Real	eCommerce	10K	-

4.2 Query Set and Error Metric

Without loss of generality, we ran every experiment on a variety of queries. All queries are in form of $\{x_1 < A < x_2 \colon x_1, x_1 \in U\}$, where A is an attribute and U is its domain. We measure the error of estimation made by histograms on the above query set by using the average of the relative error: $\frac{1}{N}\sum_{q\in Q}\frac{|\hat{C}_q - C_q|}{\hat{C}_q}$, where N is the cardinality of the query set, $\hat{C}_q$ and C_q are the actual and the estimated size of the query result set, respectively. $|\hat{C}_q - C_q|/\hat{C}_q$ represents the relative error of query q. In our experiments, we randomly generate 100 queries for each query set.

4.3 Experimental Results

Our experiments compare the two estimation algorithms proposed in Section 3. We denote them by $H1$ (in Section 3.1) and $H2$ (in Section 3.2). Without explicit explanation, the default value of the similarity threshold is 0.2 for all experiments; the default size of the real and synthetic data sets are 10,000 and 200,000 tuples; the default size of initial granularity p of partition and granularity q of repartition are 50 and 4; the default size of bucket width on similarity dimension δ_s is 0.1.

4.3.1 Effect of Data Distribution

For query estimation algorithms based on histogram, data distribution is an important factor affecting the accuracy of estimation. Fig. 2 shows the effect of data distribution to our estimation algorithms. It can be seen that for each data distribution, both two estimation algorithms can get a good estimation (relative errors are less than 40%) and algorithm $H2$ is always more accurate than algorithm $H1$.

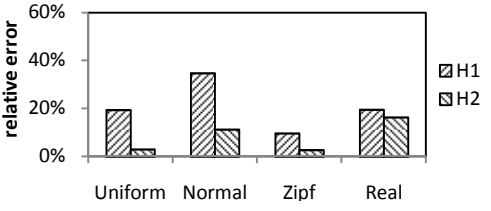

Fig. 2. Effect of data distribution

4.3.2 Effect of Data Set Size

Fig. 3 shows the effect of data set size. In this experiment, we vary the data set size by selecting the desired number of tuples from the synthetic data set T, and the data set sizes are 100K, 200K, 400K, 600K, 800K and 1000K (for eCommerce data, data set sizes are 1K, 2K, 4K, 6K, 8K and 10K). It is observed from Fig. 3 that the relative error is not sensitive to the dataset size for both two algorithms.

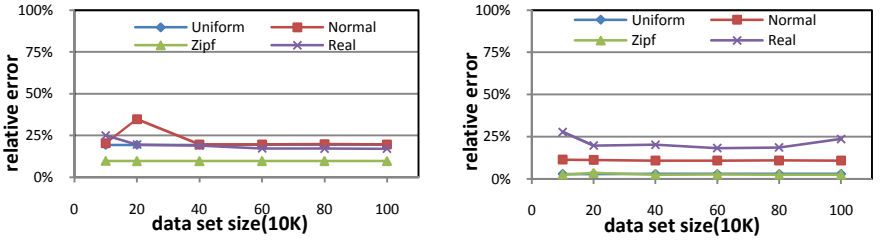

Fig. 3. Effect of data size on $H1$ and $H2$

4.3.3 Effect of Threshold

For the entity-based relational database, the threshold plays an important role in query processing. Fig. 4 shows the impact of the threshold with different thresholds from 0.1 to 0.9. For algorithm $H2$, no matter how the data is distributed, the change is not very significant. Therefore, algorithm $H2$ is relatively stable for different thresholds. However, for algorithm $H1$, the relative error decreases at first and then increases with the threshold. When the threshold is in $[0.4, 0.6]$, the relative errors is minimum. Hence the accuracy of algorithm $H1$ is related to threshold τ as mentioned in Section 3.1, and it can give accurate estimations when $\tau \in [0.4, 0.6]$.

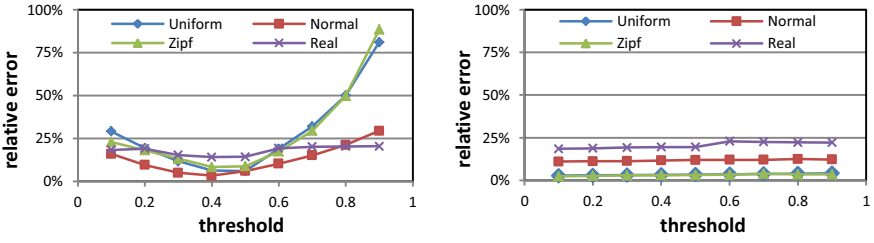

Fig. 4. Effect of threshold on $H1$ and $H2$

4.3.4 Effect of Granularity of Partition

Another important factor of our estimation algorithms is the granularity of partition. We show the effect of the initial granularity of partition p in Fig. 5, and set p at 10, 20, 40, 60, 80 and 100 respectively. We observe that there is a similar trend of the relative error both algorithm $H1$ and $H2$. As initial granularity of partition p increases, the relative error decreases. This phenomenon empirically verifies the conclusion that our approaches are unbiased when p tends to infinity.

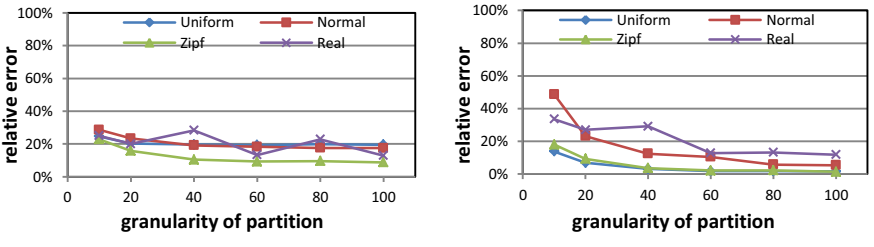

Fig. 5. Effect of granularity of partition on $H1$ and $H2$

4.3.5 Effect of Width of Query Range

In Section 3.1, we mentioned that the accuracy of the estimation using the basic histogram $H1$ is related to the width of query range and the threshold τ. Fig. 4 has showed us the effect of the threshold. In this experiment, we show the effect of the width of query range. Fig. 6 gives the experimental results. We can observe that with an increasing width of query range, the relative error has a decreasing trend for both $H1$ and $H2$, especially for $H1$, and the estimations using histogram $H2$ are more accurate. As a result, the wider query range is, the more accurate estimation is.

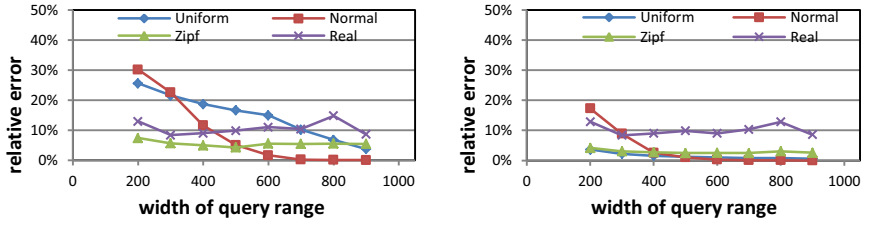

Fig. 6. Effect of width of query range on $H1$ and $H2$

5 Conclusion and Future Work

Entity-based relational database is a practical method for dirty data management. Intermediate result size estimation is crucial for the query optimization for entity-based relational database. However, traditional estimation methods cannot be applied to this problem directly. In this paper, we study this problem. To solve this problem, we propose two histogram-based methods for different form of queries and requirements. It is proven that they are unbiased. The experimental results validate the effectiveness of our algorithms, and they can indeed give good estimations for range queries. For future work, we plan to continue to study query optimization based on the cost of estimation, especially the estimation of join result size.

References

1. Batini, C., Scannapieco, M.: Data quality: concepts, methodologies and techniques. Springer (2006)
2. English, L.: Plain English on data quality: Information quality management: The next frontier. DM Review Magazine (2000)
3. Rahm, E., Do, H.H.: Data cleaning: Problems and current approaches. IEEE Data Eng. Bull. 23(4), 3–13 (2000)
4. Fuxman, A.D., Miller, R.J.: First-Order Query Rewriting for Inconsistent Databases. In: Eiter, T., Libkin, L. (eds.) ICDT 2005. LNCS, vol. 3363, pp. 337–351. Springer, Heidelberg (2005)
5. Fuxman, A., Fazli, E., Miller, R.J.: Conquer: Efficient management of inconsistent databases. In: SIGMOD, pp. 155–166 (2005)
6. Andritsos, P., Fuxman, A., Miller, R.J.: Clean answers over dirty databases: A probabilistic approach. In: ICDE, p. 30 (2006)
7. Boulos, J., Dalvi, N., Mandhani, B., Mathur, S., Re, C., Suciu, D.: MYSTIQ: a system for finding more answers by using probabilities. In: SIGMOD, pp. 891–893 (2005)
8. Widom, J.: Trio: a system for integrated management of data, accuracy, and lineage. In: CIDR, pp. 262–276 (2005)
9. Hassanzadeh, O., Miller, R.J.: Creating probabilistic databases from duplicated data. The VLDB Journal, 1141–1166 (2009)
10. Lenzerini, M.: Data integration: A theoretical perspective. In: PODS, pp. 233–246 (2002)
11. Dong, X.L., Halevy, A., Yu, C.: Data integration with uncertainty. The VLDB Journal, 469–500 (2009)
12. Benjelloun, O., Garcia-Molina, H., Menestrina, D., Whang, S.E., Su, Q., Widom, J.: Swoosh: a generic approach to entity resolution. The VLDB Journal, 255–276 (2008)
13. Li, Y., Wang, H., Gao, H.: Efficient Entity Resolution Based on Sequence Rules. In: Shen, G., Huang, X. (eds.) CSIE 2011. CCIS, vol. 152, pp. 381–388. Springer, Heidelberg (2011)
14. Ioannidis, Y.E.: The history of histograms (abridged). In: VLDB, pp. 19–30 (2003)