International Journal of Innovative
Computing, Information and Control
Volume **13**, Number **4**, August **2017**

 ISSN 1349-4198
pp. **1077–1094**

CARDINALITY ESTIMATION APPLYING MICRO SELF-TUNING HISTOGRAM

Xudong Lin[1], Xiaoning Zeng[2], Jia Liu[1] and Wei Chen[1]

[1]Department of Information Engineering
Hebei University of Environmental Engineering
No. 8, Jingang Avenue, Beidaihe Dist., Qinhuangdao 066102, P. R. China
{ fydong_xl; ning_xz; fyy_sun }@126.com

[2]College of Mathematics and Information Technology
Hebei Normal University of Science and Technology
No. 360, West Hebei Avenue, Qinhuangdao 066004, P. R. China
qhdzxn@126.com

Received December 2016; revised April 2017

Abstract. *In the cardinality estimation solutions based on multi-dimensional self-tuning histograms, periodical data scans are avoided and self-tuning histograms are constructed according to query feedback records. We call this kind of cardinality estimation solutions the reactive solutions. The existing reactive solutions are stuck with the issue of "curse of dimension". And they are unpredictable and time-consuming. To address these issues, a new reactive solution is proposed in the paper. A micro self-tuning histogram only covering the neighborhood of the new predicate is constructed, which is a beneficial attempt to improve the cardinality estimation efficiency under high dimensions, and notably alleviate the issue of "curse of dimension". Furthermore, the process of meeting a space budget is eliminated completely, which makes the whole solution reliable and dexterous.*

Keywords: Cardinality estimation, Clustering, Ward's minimum variance method, Self-tuning, Query feedback record

1. **Introduction.** In a query optimizer, cardinality estimation plays an important role in choosing optimal query plans. The first solution which applies the 1-dimensional histogram in cardinality estimation is proposed in [1]. And then, the improved solutions [2-6] are continuously proposed and the cardinality estimation technologies based on 1-dimensional histograms become relatively mature by degrees. In the mainstream relational databases, 1-dimensional histograms have been widely used to help estimate cardinality. The 1-dimensional equi-depth histogram, the 1-dimensional compressed histogram and the 1-dimensional maxdiff histogram are adopted in Oracle [7,8], DB2 [9,10] and SQL Server [11] respectively.

However, for a predicate referring to multiple attributes, multi-dimensional data summarization techniques such as multi-dimensional histograms or multi-dimensional wavelet transforms have important practical significance for cardinality estimations. Traditional multi-dimensional cardinality estimation solutions rely on data scans to summarize data. Therefore, these solutions are called the *proactive solutions* in the paper. The first proactive solution is based on the multi-dimensional equi-depth histogram [2]. And then, the improved proactive solutions are proposed continuously [12-17]. The proactive solutions proposed in [18,19] are based on the multi-dimensional wavelet transforms [20,21]. However, all of the existing proactive solutions are still in the experimental stage and no one

is actually adopted in the mainstream databases. Two serious deficiencies prevent these solutions being practical.

(1) Lots of system resources are occupied by periodical data scans and the performance of routine queries is influenced seriously.
(2) The solutions are stuck with the "curse of dimension".

The cardinality estimation solution in [22] is different from the proactive solutions. It uses the multi-dimensional self-tuning histogram to replace the proactive data summarization technologies. A multi-dimensional self-tuning histogram is constructed and maintained based on query feedback records (QFRs). We call this kind of multi-dimensional cardinality estimation solutions the *reactive solutions* in the paper. The succeeding reactive solution in [23] shows how to build low-dimensional self-tuning histograms from high-dimensional queries using the delta rule. The reactive solution in [24] improves the accuracy of a self-tuning histogram by subtilizing the granularity of QFRs. The information-theoretic principle of maximum entropy is introduced in [25] to construct a multi-dimensional self-tuning histogram which is consistent with all currently valid QFRs. [26] tries to improve the efficiency of maintaining self-tuning histograms by combining proactive histograms with QFRs. [27] uses the equi-width approach and the sparse-vector recovery based approach to maintain self-tuning histograms in the non-sparse and sparse cases respectively. [28] initializes a multi-dimensional self-tuning histogram based on subspace clustering.

Summing up the existing reactive solutions, periodical data scans are avoided but they are not yet practical due to the following common issues.

(1) Reactive solutions are still stuck with the "curse of dimension". In the existing reactive solutions, self-tuning histograms are constructed and maintained over the entire value range of the queried attributes (henceforth called *global* self-tuning histograms). As dimension increases, the bucket number of a global self-tuning histogram increases exponentially just as a proactive histogram.
(2) To construct a global self-tuning histogram, a large number of QFRs must be accumulated at a long time span. As the changes of data and workload distribution, the accumulated QFRs may become inaccurate and contradictory for each other. Therefore, the accuracy of the constructed global self-tuning histogram cannot be guaranteed.
(3) To limit the bucket number of a global self-tuning histogram, the space budgets are widely adopted in different reactive solutions. However, the limitation to bucket number leads to accuracy deterioration of cardinality estimation.

To address above issues, a new reactive solution – the *Cardinality Estimation solution applying Ward's minimum variance Method* (*CEWM*) is proposed in the paper. In CEWM, the global self-tuning histogram covering the entire value range of the queried attributes is abandoned. When a new predicate p is executed, the *Ward's minimum variance method* (*Ward method* for short) is used to find k nearest QFRs with p from the QFR warehouse. Based on the found k QFRs, a *micro* self-tuning histogram only covering the neighborhood of p is constructed to help estimate the cardinality of p. The main contributions of CEWM can be summarized as follows.

(1) The Ward method is introduced firstly to find k nearest QFRs for a new predicate. The reason that we introduce the Ward method is: for a new predicate p, there exist the executed predicates which locate in the neighborhood of p and have the relatively similar cardinalities with p. The Ward method is exactly used to find these predicates according to its function of clustering similar classes.

(2) The self-tuning histogram covering the entire value range of the queried attributes is abandoned. A micro self-tuning histogram is constructed swiftly based on a small number of QFRs of the similar executed predicates. The micro self-tuning histogram only covers the neighborhood of the new predicate but not the entire value range, which is a beneficial attempt to improve the cardinality estimation efficiency under high dimensions, and notably alleviate the issue of "curse of dimension".
(3) After the execution of each new predicate, QFR warehouse is updated by the corresponding new QFR, and the data and workload changes can be timely reflected by the micro self-tuning histogram.
(4) Due to the small scale of a micro self-tuning histogram, space budget is unnecessary to be used. The process of reducing bucket number, and the complex and cumbersome operations in the process can be eliminated completely, which make the whole solution reliable and dexterous.

The rest of the paper is organized as follows. Section 2 describes the details of finding k nearest QFRs using the Ward method. The micro histogram and its construction process are analyzed in Section 3. Based on the micro histogram, the processes of cardinality estimation for different cases are given in Section 4. The new QFR update mechanism is explained in Section 5. The results of extensive experiments are demonstrated in Section 6. Section 7 summarizes the paper and discusses future directions.

All notations used in the paper are shown in Table 1.

2. Finding k Nearest QFRs Using Ward Method.

2.1. Ward method.

Clustering [29,30] is the process of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar than those in other groups (clusters). As a hierarchical clustering method, the Ward method measures the distance between two classes based on the increment of the sum of squares of deviations (SSD). Assuming n samples are categorized into k classes $G_1, G_2, \ldots, G_k$, X_{it} is the vector composed of the relevant variables of the i sample in G_t, and n_t denotes the number of samples in G_t. $\overline{X_t}$ is the center of gravity of G_t. The SSD of G_t can be expressed as:

$$S_t = \sum_{i=1}^{n_t} \left(X_{it} - \overline{X_t}\right)^T \left(X_{it} - \overline{X_t}\right) \tag{1}$$

Assuming two classes G_p and G_q are merged into a new class G_r, SSD will increase. Based on the increment of SSD, the distance between G_p and G_q can be calculated as:

$$D_{pq} = \sqrt{S_r - S_p - S_q} \tag{2}$$

This distance is called the *Ward distance* in the paper.

2.2. Ward distance between predicates.

In CEWM, we consider each predicate as a sample in the Ward method and define the *Ward distance between predicates* firstly, and then, the k nearest QFRs with a new predicate can be found using the Ward method. For a predicate $p = (x_s \le a_s \le y_s) \wedge \cdots \wedge (x_e \le a_e \le y_e)$, the vector composed of its location variables can be expressed as:

$$X_p = (x_s, y_s, \cdots, x_e, y_e)^T \tag{3}$$

Supposing the classes G_{p_1} and G_{p_2} contain the predicate $p_1 = (x_{s1} \le a_s \le y_{s1}) \wedge \cdots \wedge (x_{e1} \le a_e \le y_{e1})$ and the predicate $p_2 = (x_{s2} \le a_s \le y_{s2}) \wedge \cdots \wedge (x_{e2} \le a_e \le y_{e2})$ respectively.

TABLE 1. Notations

basic notations	**meanings**
$\min(x)$, $\max(x)$	the minimum and the maximum of the data set x
$\|x\|$	the usual Euclidean volume of the area x when the data is real-valued; for discrete data, $\|x\|$ denotes the number of discrete points that lie in x.
r and its subscripted forms	relation
t and its subscripted forms	tuple
a and its subscripted forms	attribute
q and its subscripted forms	query
p and its subscripted forms	predicate: a predicate p has the form $(x_s \leq a_s \leq y_s) \wedge \cdots \wedge (x_e \leq a_e \leq y_e)$ where $a_s, \ldots, a_e$ are the different attributes in one relation. Given a predicate p, if it is just submitted and will be executed soon, p is called a new predicate; if p has been executed and the QFR of p has been collected, p is called an executed predicate.
h and its subscripted forms	histogram
notations related to an attribute	**meanings**
$d(a)$	the value range of a
$d(b, a)$	the value range of a which is covered by b
$d(p, a)$	the value range of a where p is true
notations related to a bucket	**meanings**
$d(b)$	the value range of b: for a bucket b of h over the attributes $a_{s+1}, \ldots, a_{s+k}$ in the relation r, $d(b) = [\min(d(b, a_{s+1})), \max(d(b, a_{s+1}))] * \cdots * [\min(d(b, a_{s+k})), \max(d(b, a_{s+k}))]$
$v(b)$	the frequency of b, i.e., the number of tuples which fall in b
notations related to a histogram	**meanings**
$d(h)$	the value range of h: a k-dimensional histogram h over the attributes $a_{s+1}, \ldots, a_{s+k}$ in the relation r is obtained by partitioning the value space $[\min(d(a_{s+1})), \max(d(a_{s+1}))] * \cdots * [\min(d(a_{s+k})), \max(d(a_{s+k}))]$ into one or more buckets and records the number of tuples falling in each bucket. $d(h) = [\min(d(a_{s+1})), \max(d(a_{s+1}))] * \cdots * [\min(d(a_{s+k})), \max(d(a_{s+k}))]$
$B(h)$	the bucket set composed of all buckets of h
notations related to a predicate	**meanings**
$d(p)$	the value range of p: for a $p = (x_s \leq a_s \leq y_s) \wedge \cdots \wedge (x_e \leq a_e \leq y_e)$, $d(p) = [\min(d(p, a_s)), \max(d(p, a_s))] * \cdots * [\min(d(p, a_e)), \max(d(p, a_e))]$
$s(p)$	The dimension of p: for a $p = (x_s \leq a_s \leq y_s) \wedge \cdots \wedge (x_e \leq a_e \leq y_e)$, $s(p)$ denotes the number of attributes $a_s, \cdots, a_e$
$qfr(p)$	the QFR of p: $qfr(p) = (p, n_p, m_p)$ where n_p is the real number of tuples satisfying p, and m_p is the executing moment of p

Simultaneously, the class $G_{p_1p_2}$ contains both p_1 and p_2. Based on (1), (2) and (3), the Ward distance between p_1 and p_2 can be defined as:

$$D_{p_1p_2} = \sqrt{S_{p_1p_2} - S_{p_1} - S_{p_2}} = \sqrt{\sum_{i=1}^{2} \left(X_{p_i} - \overline{X_{p_1p_2}}\right)^T \left(X_{p_i} - \overline{X_{p_1p_2}}\right)} \tag{4}$$

2.3. **Algorithm of finding k nearest QFRs.** Given a new predicate p_{n1} and the QFRs $qfr(p_1), \ldots, qfr(p_n)$ corresponding to the executed predicates $p_1, \ldots, p_n$, the algorithm of finding k $(1 \leq k \leq n)$ nearest QFRs is composed of the following four steps:

Step 1: Categorize p_{n1} and $p_1, \ldots, p_n$ into $n+1$ classes $G_{p_{n1}}, G_{p_1}, \ldots, G_{p_n}$ and each class only contains one predicate;

Step 2: Merge $G_{p_{n1}}$ with every one of $G_{p_1}, \ldots, G_{p_n}$ respectively and calculate the Ward distances $D_{p_{n1}p_x}$ for $x = 1, \ldots, n$ according to (4).

Step 3: Sort $D_{p_{n1}p_x}$ for $x = 1, \ldots, n$ to get the ascending QFR sequence.

Step 4: Configure k value. The detail can be found in Section 3.4. The top k QFRs in the ascending QFR sequence are the k nearest QFRs with p_{n1}.

The pseudo-codes of above steps are shown as Algorithm 1.

Algorithm 1: finding k nearest QFRs with the new predicate p_{n1}

```
fkqfr(p_n1, p[1 ··· n])
1 G_pn1 ← p_n1
2 for (each i ∈ [1 ··· n]) do
3       G_p[i] ← p[i]
4    wd[i] ← calWardDis(G_pn1, G_p[i]) //calculating Ward distances
5 loc ← sort(wd[1], ..., wd[n]) //sorting Ward distances
6 k ← configK(p_n1, p[1 ··· n], loc[1 ··· n]) //configuring k, elaborated in Algorithm 2
7 return loc[1 ··· k]
```

For example, a 2-dimensional new predicate p_{n1} and ten executed predicates $p_1, \ldots, p_{10}$ are shown in Figure 1. The new predicate is shown as a grey rectangle and the rectangles

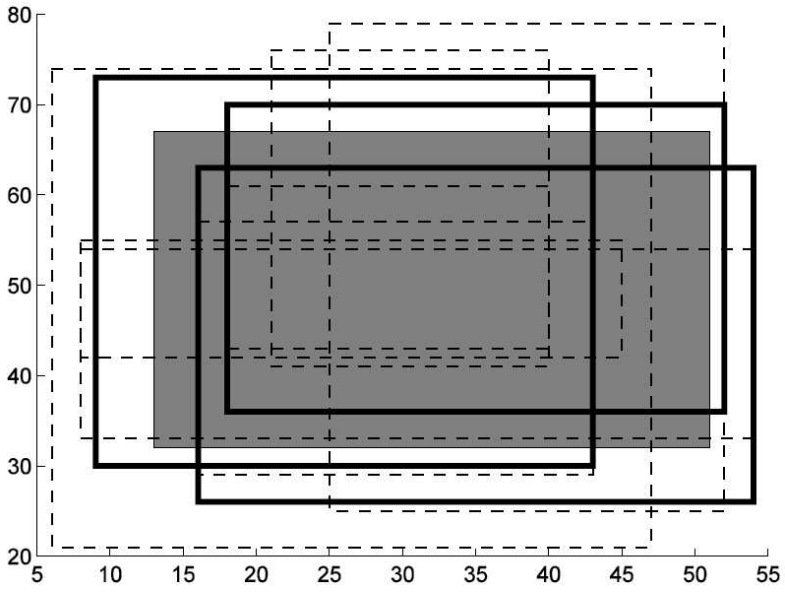

FIGURE 1. A new predicate and ten executed predicates

with dashed borders denote the executed predicates. Configure $k = 3$, the executed predicates corresponding to the k nearest QFRs with p_{n1} are shown as the rectangles with bold solid borders. They can be efficiently obtained by Algorithm 1.

2.4. **The choice of *k* value.** Using Algorithm 1, the k nearest QFRs with a new predicate can be found. However, k value must be chosen carefully. For an actual database, especially an OLTP system, the query workload often shows a certain skewed distribution feature where some tuples are frequently queried but many tuples are not [3,31]. Figure 2 shows a skewed query workload example in a 2-dimensional space. Each predicate is denoted by a rectangle. The area surrounded by a rectangle with bold solid borders is more frequently queried than the other areas.

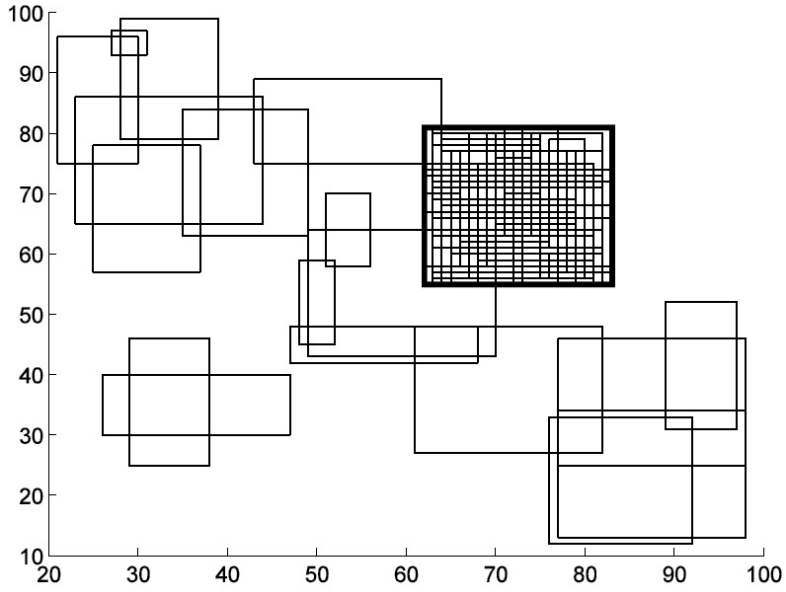

FIGURE 2. A skewed query workload example

For a new predicate locating in the frequently queried areas, it is common that some executed predicates can be found whose value range union can cover the value range of the new predicate. In this case, the choice of k value should correspond to a minimum necessary predicate set. For the new predicates locating in the infrequently queried areas, we configure an upper limit of k value to avoid appearing a micro histogram with large scale. In summary, for a new predicate p, the k value can be chosen as follows:

$$k = \min(C_p, UL) \tag{5}$$

In (5), C_p is a positive integer and can be various for different predicates. And UL is the upper limit of k value which can be adjusted in the running time according to the real-time query workload. C_p and UL fulfill $C_p < UL$.

When $k = \min(C_p, UL) = C_p$, the top C_p QFRs in the ascending QFR sequence obtained in Algorithm 1 fulfill: (1) the value range union of the corresponding C_p executed predicates can cover $d(p)$; (2) the value range union of the corresponding $C_p - 1$ executed predicates cannot cover $d(p)$. When $k = \min(C_p, UL) = UL$, the value range union of the $UL - 1$ executed predicates corresponding to the top $UL - 1$ QFRs in the ascending QFR sequence obtained in Algorithm 1 cannot cover $d(p)$.

For a new predicate p_{n1}, the drilling hole operation [25] will be executed to calculate k value. The pseudo-codes of configuring k value are shown as Algorithm 2.

Algorithm 2: configuring k value for the new predicate p_{n1}

configK(p_{n1}, $p[1 \cdots n]$, $loc[1 \cdots n]$)
1 $k \leftarrow UL$ // UL is the upper limit of k
2 $ucas \leftarrow ucas \cup uca_{n1}$ //the array $ucas$ stores all uncovered areas inside $d(p_{n1})$ according to the k executed predicates. It is initialized with the area uca_{n1} covering the whole $d(p_{n1})$.
3 **for** (each $i \in [1 \cdots UL-1]$) **do**
4 **if** ($|ucas| == 0$)
5 $k \leftarrow i$
6 **break**
7 **else**
8 $tempUcas \leftarrow \emptyset$
9 **for** (each $uca \in ucas$) **do**
10 **if** ($d_{uca} \subseteq d_{p[loc[i]]}$)
11 $ucas \leftarrow ucas$ – uca
12 **else if** ($d_{uca} \cap d_{p[loc[i]]} \neq \emptyset$)
13 $tempUcas \leftarrow tempUcas \cup holeDrilling$ (p[loc[i]], uca)
14 $ucas \leftarrow ucas$ – uca
15 $ucas \leftarrow ucas \cup tempUcas$
16 **return** k

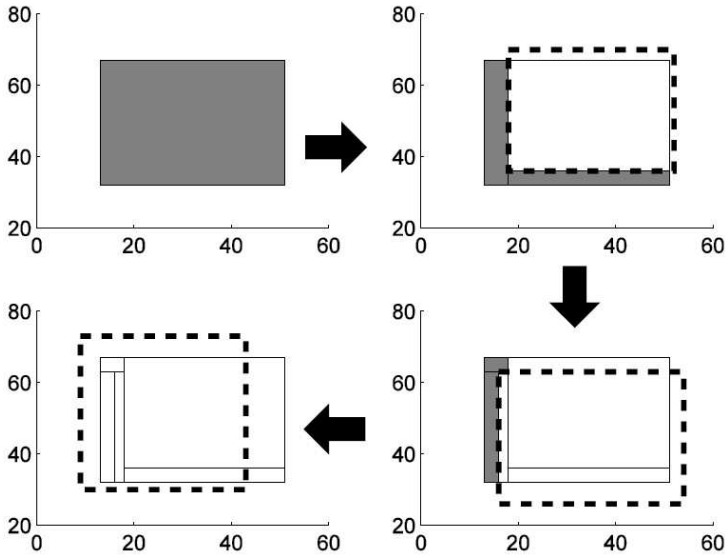

FIGURE 3. Process of configuring k value

For the new predicate shown in Figure 1, the process of configuring k value is shown in Figure 3. The uncovered areas inside the value range of the new predicate are filled with grey. After the drilling hole operations according to 3 executed predicates, no any uncovered areas can be found. Therefore, the k value can be configured as 3.

3. **Constructing Micro Histogram.** A micro histogram is constructed for the predicate p based on its k nearest QFRs to estimate its cardinality.

3.1. **Micro histogram and global histogram.** A micro histogram is a histogram only covering a local of the value ranges of the queried attributes. In contrast, a histogram covering the entire value ranges of the queried attributes is called a global histogram.

The reason that we adopt the micro histogram but not the global histogram in CEWM is: 1) the cardinality of a predicate is only decided by the data distribution of its neighborhood but not the entire value ranges of the queried attributes; 2) it is more efficient to construct and maintain a micro histogram because the number of QFRs participating in the construction is very limited; 3) the changes of the underlying data can be embedded into a micro histogram in a more timely manner.

3.2. **Constructing micro histogram.** For a multi-dimensional micro histogram h constructed over a certain local of the value ranges of the attributes $a_{s+1}, \ldots, a_{s+k}$ in the relation r, each bucket $b_i \in B(h)$ covers a hyper rectangle with two constant boundaries in each dimension. Inside each hyper rectangle, there may be some mutually disjoint sub hyper rectangles which are covered by the other buckets $b_{i1}, \ldots, b_{ij}$. We say $b_{i1}, \ldots, b_{ij}$ are the children of b_i and all buckets in the histogram h compose a tree structure. The value range of b_i can be calculated as follows:

$$d(b_i) = d\left(R_{b_i}\right) - \cup_{x=1}^{j} d(b_{ix}) \tag{6}$$

Here $d(R_{b_i})$ denotes the entire value range of the hyper rectangle covered by b_i. An example of the buckets in a multi-dimensional micro histogram and the corresponding bucket tree are shown in Figure 4.

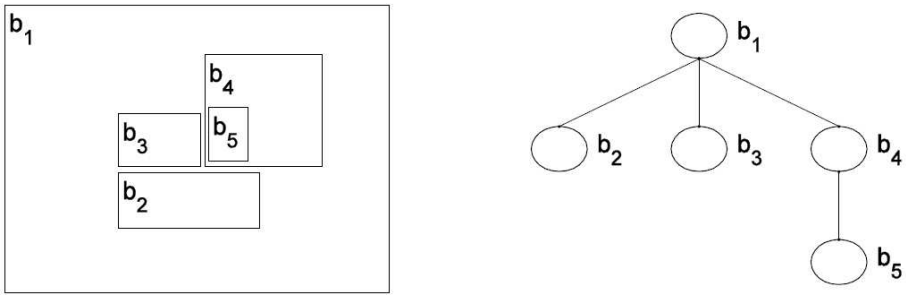

FIGURE 4. A micro histogram and the corresponding bucket tree

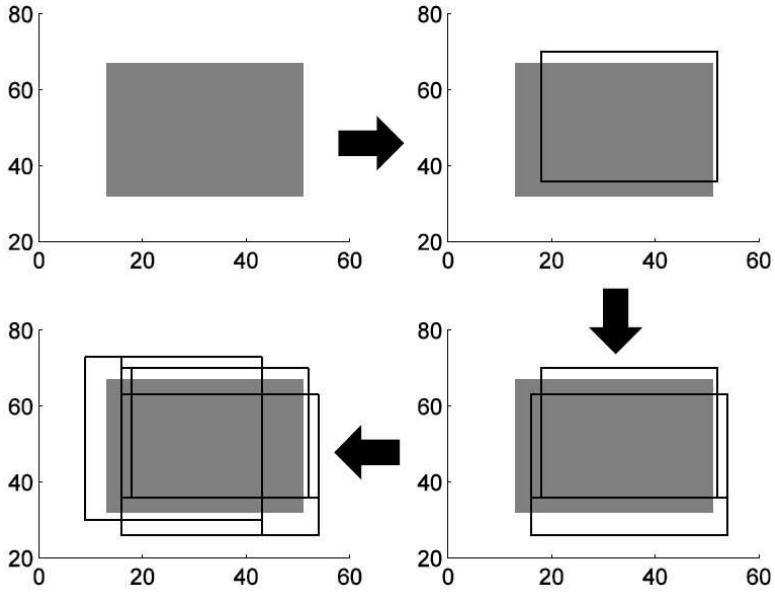

FIGURE 5. Process of generating all buckets in a micro histogram

For each new predicate p_{n1} and its k nearest QFRs $qfr(p_1), \ldots, qfr(p_k)$, a micro histogram h is initialized with one bucket covering $d(p_1)$. And then, for each executed predicate p_i $(i = 2, \ldots, k)$, CEWM tries to find a bucket $b \in B(h)$ which satisfies $d(b) = d(p_i)$. If there is no such bucket, the drilling hole operation will be executed to generate the new buckets. For the new predicate shown in Figure 1, the process of generating all buckets in the micro histogram is shown in Figure 5, where the new predicate is shown as a grey rectangle.

Once the k executed predicates p_i $(i = 1, \ldots, k)$ are processed and all buckets of h are generated, the iterative scaling (IS) algorithm [32] will be loaded to calculate the frequencies of all buckets. The pseudo-codes of constructing a micro histogram are shown as Algorithm 3.

Algorithm 3: constructing the micro histogram h for the new predicate p_{n1}

constLh(p_{n1}, $p[1 \cdots n]$, $loc[1 \cdots k]$)
1 $T_{Bh} \leftarrow T_{Bh} \cup b_0$ //The bucket tree T_{Bh} is initialized with the bucket b_0 covering the whole $d(p_{loc[1]})$.
2 **for** (each $i \in loc[2 \cdots k]$) **do**
3 **if** (!*srhEqu*(T_{Bh}, $p[i]$)) //searching the bucket b' satisfying $d(b') = d(p_i)$
4 $T_{temp} \leftarrow \emptyset$
5 **for** (each $(b \in T_{Bh})\&\&(d_b \cap d_{p[i]} \neq \emptyset)$) **do**
6 $T_{temp} \leftarrow T_{temp} \cup$ *holeDrilling* $(p[i], b)$
7 $T_{Bh} \leftarrow T_{Bh} - b$
8 $T_{Bh} \leftarrow T_{Bh} \cup T_{temp}$
9 $h \leftarrow$ is $(T_{Bh}, n(p[loc[1]]) \cdots n(p[loc[k]])$ //executing the IS algorithm

4. **Cardinality Estimation.** Assuming the micro histogram h is constructed to estimate the cardinality of a new predicate p. For a bucket $b_i \in B(h)$ satisfying $d(p) \cap d(b_i) \neq \emptyset$, the number of tuples which fall in b_i and satisfy p can be estimated as:

$$est_{b_i}(p) = v(b_i) * |d(b_i) \cap d(p)|/|d(b_i)| \tag{7}$$

Based on (7), the cardinality of p can be estimated in three different cases:

Case 1: If there exists the bucket set $B'(h) \subseteq B(h)$ satisfying $d(p) \subseteq \cup_{b_i \in B'(h)} d(b_i)$, and for each $b_i \in B'(h)$, $d(p) \cap d(b_i) \neq \emptyset$, the cardinality of p is:

$$est(p) = \sum\nolimits_{b_i \in B'(h)} est_{b_i}(p) \tag{8}$$

Case 2: If no bucket set $B'(h) \subseteq B(h)$ satisfying $d(p) \subseteq \cup_{b_i \in B'(h)} d(b_i)$ can be found, but there exists a bucket set $B''(h) \subseteq B(h)$ satisfying $d(p) \cap d(b_i) \neq \emptyset$ for each $b_i \in B''(h)$, the cardinality of p is:

$$est(p) = \sum\nolimits_{b_i \in B''(h)} est_{b_i}(p) * |d(p)| \Big/ \sum\nolimits_{b_i \in B''(h)} |d(b_i) \cap d(p)| \tag{9}$$

Case 3: If no any bucket $b_i \in B(h)$ satisfies $d(p) \cap d(b_i) \neq \emptyset$, the cardinality of p is:

$$est(p) = \sum\nolimits_{b_i \in B(h)} v(b_i) * |d(p)| \Big/ \sum\nolimits_{b_i \in B(h)} |d(b_i)| \tag{10}$$

For example, the cardinalities of the new predicates p_{n1}, p_{n2} and p_{n3} in Figure 6 can be calculated using (8), (9) and (10) respectively.

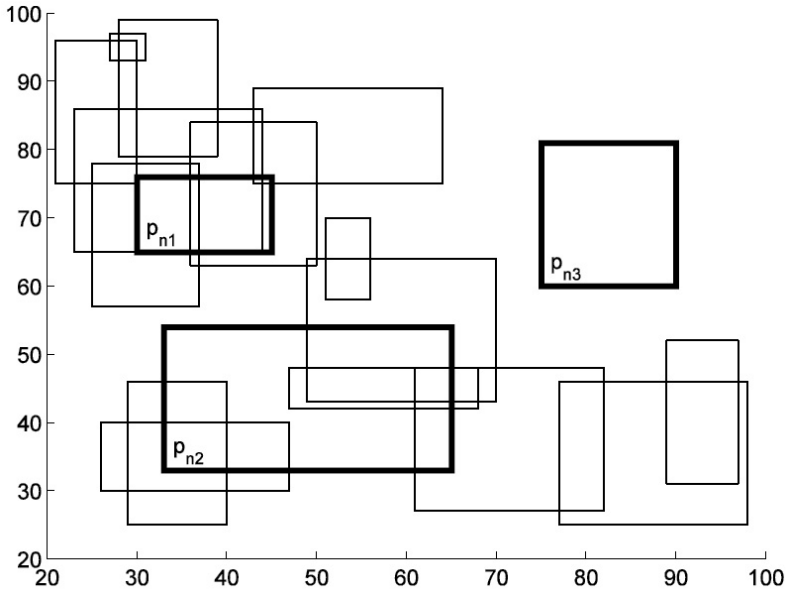

FIGURE 6. Cardinality estimations in different cases

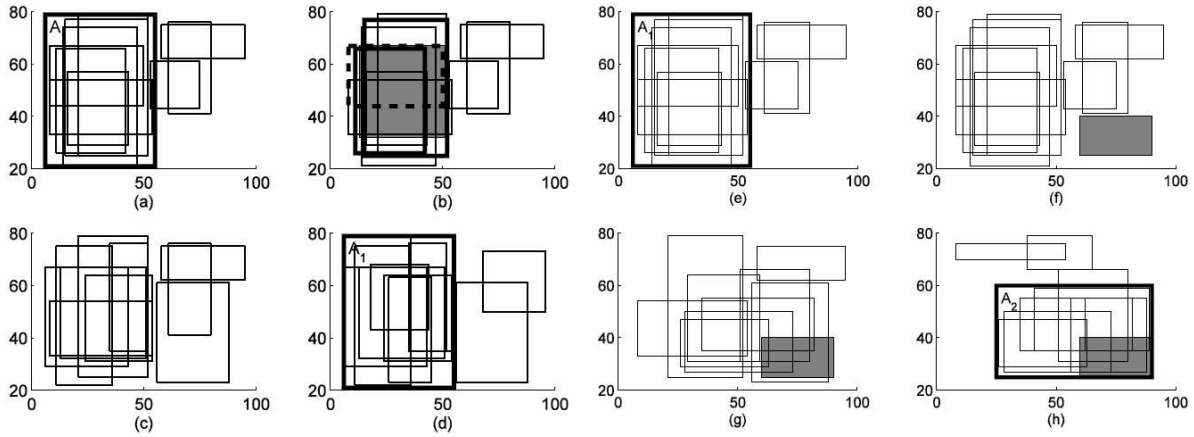

FIGURE 7. QFR updates without query workload changes

5. **QFR Update Mechanism.** We adopt the following QFR update mechanism in CEWM: *for a new predicate p and its k nearest QFRs, qfr(p) will replace the oldest one of the k nearest QFRs after p is executed.*

Assuming the value ranges of ten successively executed predicates p_1 to p_{10} are shown in Figure 7(a). We can observe the skewed distribution feature of p_1 to p_{10} and most of them locate in A_1 area. The QFRs $qfr(p_1)$ to $qfr(p_{10})$ are stored into a QFR warehouse after the executions of p_1 to p_{10}.

Subsequently, ten new predicates p_{n1} to p_{n10} which have the same distribution feature with p_1 to p_{10} are executed successively. Based on the QFR update mechanism in CEWM, the changes of the QFR warehouse are shown in Figure 7(b) to Figure 7(d) respectively (Some medium statuses are omitted).

We also assume the other ten successively executed new predicates p_{n11} to p_{n20} have the different distribution features with p_1 to p_{10}. Based on the QFR update mechanism in CEWM, the changes of the QFR warehouse are shown in Figure 7(e) to Figure 7(h) respectively (Some medium statuses are omitted).

The QFR update mechanism in CEWM can make a micro histogram match not only the changes of underlying data but also the changes of query workload distribution.

6. Experiments.

The experiments are performed on a 3.2GHz Intel CPU machine running Windows 7 sp1, with 4GB memory and 1TB hard disk. Before analyzing the experimental results, we describe the experimental settings firstly.

6.1. Experimental settings.

6.1.1. *Data sets.* To test CEWM comprehensively, a real data set [33] and the TPC-H benchmark [34] with scale factor of 1 are used for the experiments. The distribution of the former shows skewed feature and the latter is a uniform data set. The details of the two data sets are described as follows.

Real data set (denoted by ds_1): The real data set contains the census data of U.S. in 1990 which come from the UCI Machine Learning Repository. A relation *census_1990* is created in the commercial relation database Oracle 12c to store the data set which contains 2,458,285 tuples. All experiments are carried out over the attributes *income1* and *income2*.

TPC-H benchmark (denoted by ds_2): A relation *order* is created in the commercial relation database Oracle 12c to store 1,500,000 tuples which are generated by the DBGEN program. All experiments are carried out over the attributes *o_orderdate* and *o_orderId*.

6.1.2. *Query workloads.* In our experiments, two query workload models qw_1 and qw_2 which follow the Zipfian distribution and the Gaussian distribution are adopted. Based on the feature of the two distributions, they can be considered as the approximate descriptions of the skewed query workload in a practical database.

6.1.3. *Metrics.* Firstly, we can define $re(p)$, the relative error of a predicate p using (11):

$$re(p) = \frac{abs(n(p) - est(p))}{n(p)} \tag{11}$$

Based on relative errors, we define the relative accuracy rate, $rar(ce)$, of a cardinality estimation solution ce as the criterion to measure the accuracy of a cardinality estimation solution:

$$rar(ce) = \frac{cn_s(ce)}{tn(ce)} \tag{12}$$

where ce denotes a cardinality estimation solution, $cn_s(ce)$ denotes the number of predicates whose relative errors are lower than s, and $tn(ce)$ denotes the total number of predicates. In our experiments, we configure $s = 0.2$ and consider a predicate whose relative error is lower than 0.2 as a correctly estimated predicate.

6.1.4. *Programs.* In our experiments, the comparison solutions include CEWM proposed in the paper and the representative reactive solution ISOMER.

CEWM and ISOMER are realized under JDK 1.6.0_10. For CEWM, the initial capacity of QFR warehouse and the upper limit of k value are configured as 300 and 10 respectively. For ISOMER, the space budget of histogram affects the experimental results remarkably. Therefore, we compare two kinds of ISOMER solutions – the *ISO2* solution with 200 space budgets of histogram, and the *ISO3* solution with 300 space budgets of histogram.

6.2. **Static experiments.** At the preparation stage of each static experiment based on one data set and one query workload, 300 *training predicates* will be executed and the corresponding QFRs will be stored into the QFR warehouse for CEWM. And for ISO2 and ISO3, the initial histograms with about 200 and 300 buckets will be constructed using 150 and 200 training predicates.

And then, in the formal experimental stage, 1,000 *validation predicates* with the same distribution feature as the training predicates will be executed using different solutions. During the execution of the 1,000 validation predicates, for each 100 ones, the relative accuracy rate and the overall execution time will be recorded for each solution.

The results of the static experiments based on ds_1 and qw_1 are shown in Figures 8(a) and 8(b). From Figure 8(a), we can see that CEWM shows excellent accuracy of cardinality estimation and the relative accuracy rates of CEWM are always higher than 80 percent.

For ISOMER, the accuracy of ISO3 is better than the one of ISO2 due to the improved space budget of histogram, but the overall accuracy level of ISOMER is about 20 to 30 percent lower than CEWM.

From Figure 8(b), we can also see the superiority of CEWM in efficiency. CEWM can finish the cardinality estimations of each 100 predicates within 10 seconds in general.

However, for ISOMER, the time costs are 10 to 50 multiples of CEWM due to the periodical reconstructions of global histograms. Furthermore, as the space budgets of histograms increase, the efficiency of ISOMER deteriorates rapidly. Although the relative accuracy rates of the ISO3 solution show about 10 percent improvements compared with the ISO2 solution, the efficiencies drop 70 percent averagely.

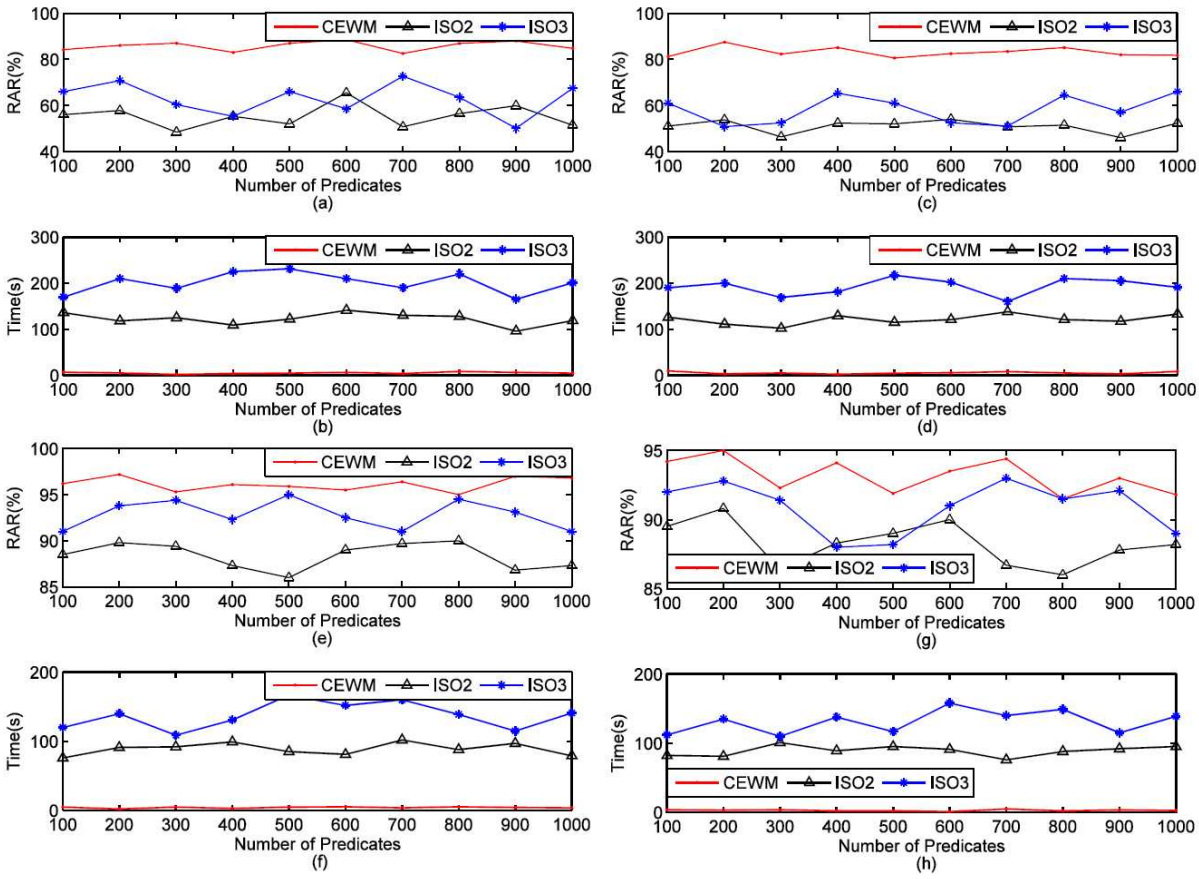

FIGURE 8. Static experiments

Figure 8(c) and Figure 8(d) show the results of the static experiments based on ds_1 and qw_2. Using a query workload following the 2-dimensional Gaussian distributions, the changing tendencies of the relative accuracy rate and the execution time are similar to the ones in Figure 8(a) and Figure 8(b). CEWM is still accurate, stable and efficient. And the fluctuation of ISOMER is still obvious.

Figure 8(e) and Figure 8(f) show the results of the static experiments based on ds_2 and qw_1. When the experiments are carried out over a uniform data set, the relative accuracy rates of all solutions improve to different degrees. The overall accuracy level of CEWM is still higher than ISOMER. Most of the relative accuracy rates of CEWM exceed 95 percent. And the relative accuracy rates of the ISO3 solution are between 90 percent and 95 percent.

Over the uniform underlying data, CEWM can always provide the accurate cardinality estimations with only 5 percent to 10 percent time costs of the ISO2 solution.

The results of the static experiments based on ds_2 and qw_2 are shown in Figure 8(g) and Figure 8(h). The tiny difference of skewness between ds_1 and ds_2 is the main reason leading to the slight deterioration of CEWM in accuracy and stability over ds_2.

Compared with ISOMER, CEWM can finish cardinality estimations more accurately in much shorter time. However, the practicability of a solution must be further tested based on the dynamic experiments.

6.3. **Dynamic experiments.** The preparation stage and the formal stage of each dynamic experiment are similar to the static experiment, and 300 training predicates and 1,000 validation predicates will be executed in the two stages respectively. The differences between the static experiment and the dynamic experiment are the changes of underlying data and query workloads.

In each dynamic experiment, as the main process of each cardinality estimation solution is being executed, another data updating process is running simultaneously. For the data set ds_1, the data updating process contains two refresh functions *INS* and *DEL*, which can insert 10 percent new tuples into the relation *census*_1990 and delete 10 percent old tuples from the relation *census*_1990 respectively. Both *INS* and *DEL* will be executed once before each 100 of the 1,000 validation predicates are executed, which can ensure at least 20 percent of the data in ds_1 can be updated. For the data set ds_2, the refresh functions *RF1* and *RF2* which are defined in the TPC-H benchmark will be used to finish the update of the underlying data. Before each 100 of the 1,000 validation predicates are executed, *RF1* and *RF2* will be executed 100 times continuously to ensure at least 20 percent of the data in ds_2 can be updated.

As the underlying data are updated by the refresh functions, the query workloads are also changed. For the data set ds_1, each 100 of the 1,000 validation predicates will satisfy the 2-dimensional Zipfian distribution with the new centers for the 5 mutually disjoint parts. And for the data set ds_2, each 100 of the 1,000 validation predicates will satisfy the superposition of 2-dimensional Gaussian distributions with 3 new median pairs.

The results of the dynamic experiments based on ds_1 and qw_1 are shown in Figure 9(a) and Figure 9(b). Compared with the results of the corresponding static experiments in Figure 8(a) and Figure 8(b), we can observe the relative accuracy rates of all solutions decline due to the changes of underlying data and query workloads. For CEWM, the average of the relative accuracy rates declines from 85.8 percent to 75.9 percent. And the ones of the ISO2 solution and the ISO3 solution also decline from 55.4 percent and 63.1 percent to 43.3 percent and 46.6 percent respectively. The difference between the accuracies of CEWM and ISOMER becomes much larger. Although the high time cost

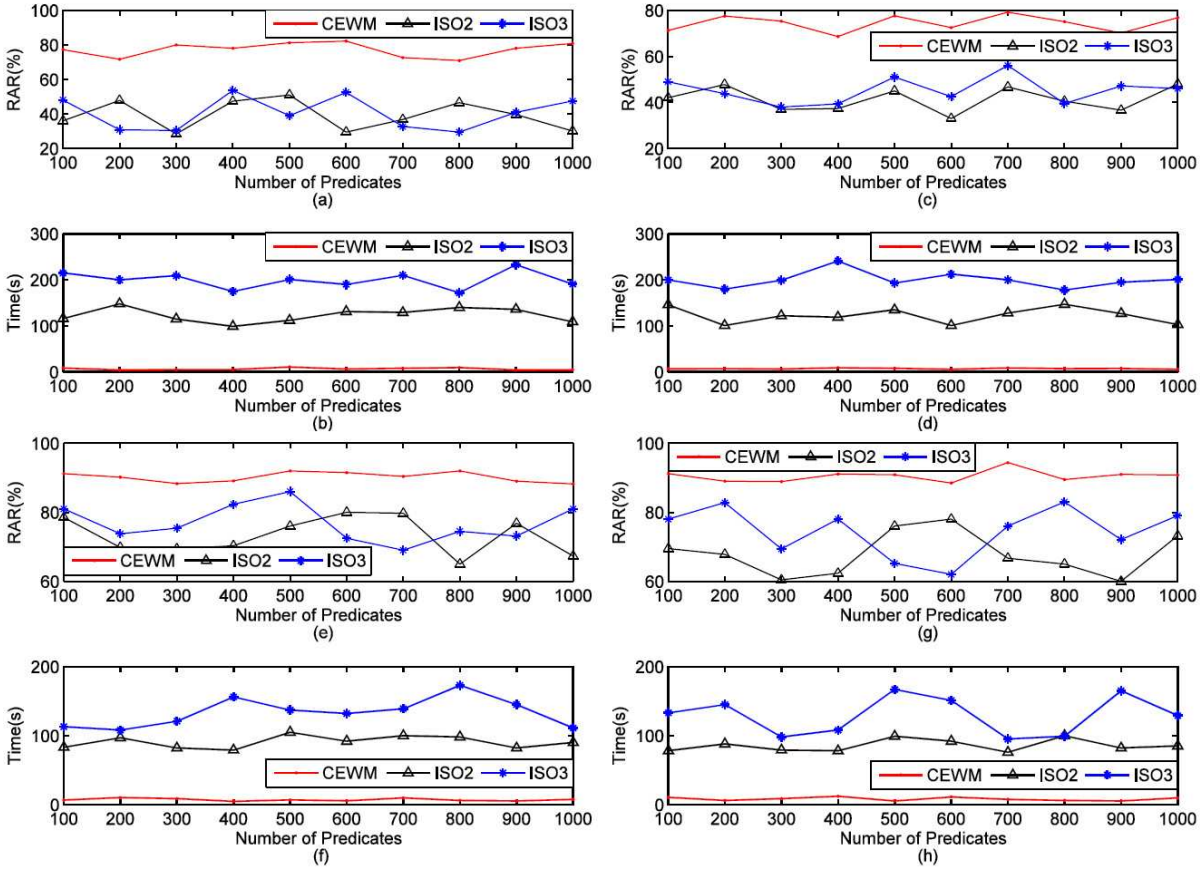

FIGURE 9. Dynamic experiments

is paid out, the ISO3 solution can only provide accurate cardinality estimations for fewer than half of the validation predicates under the changing data and query workloads.

The difference between the maximum and the minimum of the relative accuracy rates of CEWM increases from 5.9 percent to 11.3 percent, which show that the changing underlying data and query workloads bring larger fluctuations. However, compared with ISOMER, CEWM is still a much more stable solution. Furthermore, CEWM can always provide accurate cardinality estimations for more than 70 percent validation predicates under the changing data and query workloads.

Besides accuracy, the efficiency superiority of CEWM can still be observed from Figure 9(b). The changes of underlying data and query workloads have little effect on the efficiency of CEWM and ISOMER. CEWM can always provide relatively accurate cardinality estimations with only 5 percent to 10 percent time costs of the ISO2 solution.

Apparently, CEWM is more adaptive to the changes than ISOMER, which owe to the micro histogram and the QFR update mechanism adopted in CEWM.

Figure 9(c) and Figure 9(d) show the results of the dynamic experiments based on ds_1 and qw_2. Using a query workload following the 2-dimensional Gaussian distribution, the experimental results are similar with the ones in Figure 9(a) and Figure 9(b).

Figure 9(e) and Figure 9(f) show the results of the dynamic experiments based on ds_2 and qw_1. Most relative accuracy rates of CEWM are higher than 90 percent. The instability is always a serious deficiency of ISOMER. For the 600th to the 700th predicates, the relative accuracy rate of the ISO3 solution is only 69 percent, but for the 400th to the 500th predicates, the value is 86 percent. ISOMER cannot stably provide accurate

cardinality estimation even for the uniform underlying data. From Figure 9(f), we can still observe the superiority of CEWM in efficiency.

The results of the dynamic experiments based on ds_2 and qw_2 in Figure 9(g) and Figure 9(h) show more fluctuations than the ones shown in Figure 9(e) and Figure 9(f).

In summary, CEWM can adapt to the changes of the underlying data and the query workloads much better than ISOMER.

6.4. **Parameter influence experiments.** During the execution of CEWM, two parameters can be configured freely, i.e., the initial capacity of the QFR warehouse and the upper limit of k value. Whether the configurations of the two parameters influence the performance of CEWM should be tested by the corresponding experiments.

To test the influence of the initial capacity of the QFR warehouse, we configure it as 100, 300, 500, 700 and 900 (QFRs) respectively. And then the results of the dynamic experiments based on ds_1 and qw_1 are shown in Figure 10(a) and Figure 10(b), where ic denotes the initial capacity of the QFR warehouse. When the initial capacity of the QFR warehouse equals 100, the relative accuracy rate of CEWM is a little lower. When the initial capacity of the QFR warehouse increases to 300 or more, we cannot observe the apparent differences between the relative accuracy rates corresponding to the different initial capacities of the QFR warehouse. By analyzing the detailed cardinality estimation result of each validation predicate, we conclude that 100 is not an enough initial capacity of the QFR warehouse for CEWM. However, when the initial capacity of the QFR warehouse increases to 300, the accuracy of cardinality estimation can be fully guaranteed.

From Figure 10(b), we know the efficiency of CEWM cannot be influenced by the initial capacity of the QFR warehouse remarkably.

Figure 10(c) and Figure 10(d) show the experimental results of the initial capacity of the QFR warehouse based on ds_1 and qw_2. No remarkable difference can be observed from the results with the ones in Figure 10(a) and Figure 10(b).

We also carry out the experiments aiming at the influence of the upper limit of k value on CEWM. The results of the dynamic experiments based on ds_1 and qw_1 are shown in Figure 11(a) and Figure 11(b). And Figure 11(c) and Figure 11(d) show the results of the dynamic experiments based on ds_1 and qw_2. In Figure 11, the upper limit of k value is configured as 10, 20, 30, 40 and 50 respectively. We cannot see obvious difference from the relative accuracy rates under different upper limits of k value.

However, from Figure 11(b) and Figure 11(d), we can see the configuration of different upper limits of k value mainly influence CEWM on efficiency. As the increase of the upper

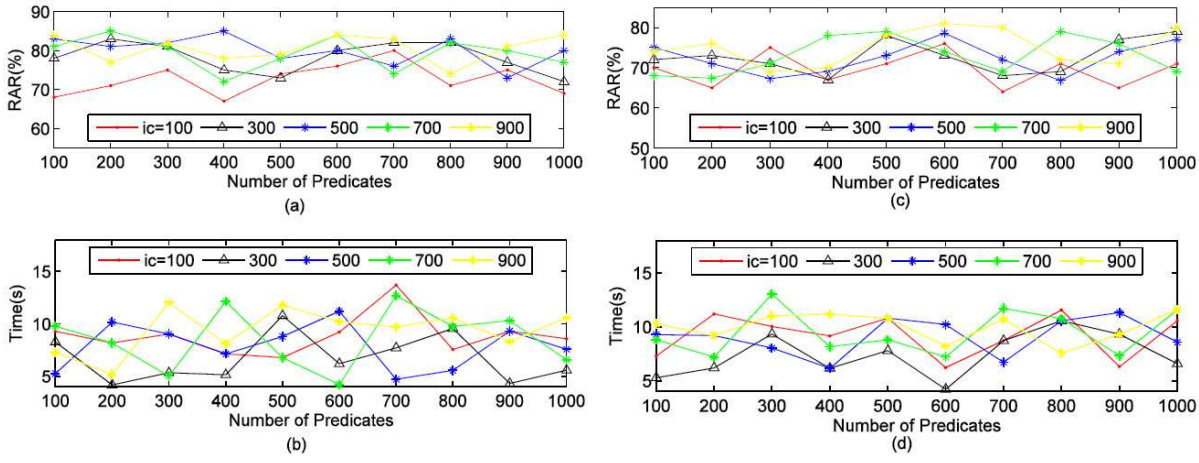

FIGURE 10. Experiments for initial capacity

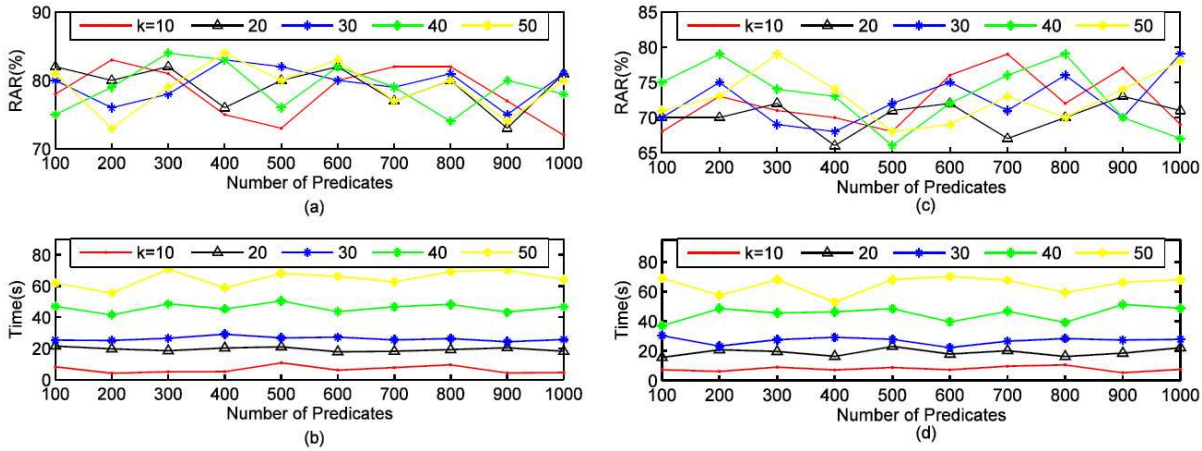

FIGURE 11. Experiments for different k

limit of k value, the average time of cardinality estimation of each predicate increases from 90ms to 650ms. Apparently, the increments of time cost originate from the cardinality estimations of the predicates locating in the infrequently queried areas. However, as we know, the overall accuracy of CEWM is much more decided by the cardinality estimations for the predicates locating in the frequently queried areas but not the ones in the infrequently queried areas. Therefore, it is unnecessary to adopt an upper limit of k value more than 10.

In summary, the performance of CEWM is not seriously influenced by the initial capacity of the QFR warehouse and the upper limit of k value. In general, CEWM can work well under the initial capacity of the QFR warehouse 300, and the upper limit of k value 10.

7. **Conclusion.** In the paper, a new cardinality estimation solution using micro self-tuning histograms is proposed. The Ward method is introduced to find k nearest QFRs for a new predicate, and the micro self-tuning histogram is constructed based on the k nearest QFRs to alleviate the issue of "curse of dimension" and improve the efficiency of the solution. The data and workload changes can be timely reflected by the micro self-tuning histogram. The complex and cumbersome operations in the process of meeting a space budget are eliminated completely, which make the whole solution reliable and dexterous. Extensive comparison experiments have shown the outstanding performance of our solution.

In the future, we will use the application framework in the paper to improve the cardinality estimation of join predicates which are related to multiple attributes in different relations.

Acknowledgment. This work is supported in part by the Research Funds from the Science and Technology Plan of Hebei Province under Grant No. 15210334, No. 16210348, and the Young Science and Technology Research Foundation for the Colleges and Universities of Hebei Province under Grant No. QN2015133. We would like to thank the anonymous reviewers for their constructive suggestions.

REFERENCES

[1] R. P. Kooi, *The Optimization of Queries in Relational Databases*, 1980.

[2] M. Muralikrishna and D. J. Dewitt, Equi-depth histograms for estimating selectivity factors for multi-dimensional queries, *ACM SIGMOD International Conference on Management of Data*, Chicago, IL, pp.28-36, 1988.
[3] Y. E. Ioannidis and V. Poosala, Balancing histogram optimality and practicality for query result size estimation, *Acm Sigmod Record*, vol.24, no.2, pp.233-244, 1995.
[4] G. Piatetskyshapiro and C. Connell, Accurate estimation of the number of tuples satisfying a condition, *Sigmod'84, Proc. of Meeting*, Boston, MA, pp.256-276, 1984.
[5] V. Poosala, P. J. Haas, Y. E. Ioannidis et al., Improved histograms for selectivity estimation of range predicates, *Acm Sigmod Record*, vol.25, no.2, pp.294-305, 1999.
[6] H. To, K. Chiang and C. Shahabi, Entropy-based histograms for selectivity estimation, *International Conference on Information and Knowledge Management*, pp.1939-1948, 2013.
[7] L. Ashdown, M. Colgan and T. Kyte, *Oracle Database SQL Tuning Guide 12c Release 1 (12.1), E49106-05*, http://docs.oracle.com/database/121/TGSQL/E49106-05.pdf, 2014.
[8] D. R. Augustyn, *Applying Advanced Methods of Query Selectivity Estimation in Oracle DBMS*, Man-Machine Interactions, Springer Berlin Heidelberg, pp.585-593, 2009.
[9] IBM Corp., *DB2 Version 10.5 for Linux, UNIX and Windows English manuals*, http://public.dhe.ibm.com/ps/products db2/info/vr105/pdf/en_US/db2_v105_books_en_US.zip, 2014.
[10] M. Stillger, G. M. Lohman, V. Markl et al., LEO – DB2's LEarning optimizer, *International Conference on Very Large Data Bases*, pp.19-28, 2001.
[11] S. Agrawal, S. Chaudhuri, L. Kollar et al., Database tuning advisor for Microsoft SQL Server 2005, *ACM SIGMOD International Conference on Management of Data*, Baltimore, MO, USA, pp.1110-1121, 2004.
[12] V. Poosala and Y. E. Ioannidis, Selectivity estimation without the attribute value independence assumption, *VLDB*, pp.486-495, 1997.
[13] D. Gunopulos, G. Kollios, V. J. Tsotras et al., Approximating multi-dimensional aggregate range queries over real attributes, *Acm Sigmod Record*, vol.29, no.2, pp.463-474, 2001.
[14] A. Deshpande, M. Garofalakis and R. Rastogi, Independence is good: Dependency-based histogram synopses for high-dimensional data, *ACM SIGMOD International Conference on Management of Data*, pp.199-210, 2001.
[15] N. Thaper, S. Guha, P. Indyk et al., Dynamic multidimensional histograms, *ACM SIGMOD International Conference on Management of Data*, Madison, WI, pp.428-439, 2002.
[16] H. Wang and K. C. Sevcik, A multi-dimensional histogram for selectivity estimation and fast approximate query answering, *Conference of the Centre for Advanced Studies on Collaborative Research*, Toronto, Ontario, Canada, pp.328-342, 2003.
[17] L. Baltrunas, A. Mazeika and M. Bohlen, Multi-dimensional histograms with tight bounds for the error, *The 10th International Database Engineering and Applications Symposium*, pp.105-112, 2007.
[18] J. S. Vitter and M. Wang, Approximate computation of multidimensional aggregates of sparse data using wavelets, *Acm Sigmod Record*, vol.28, no.2, pp.193-204, 1999.
[19] K. Chakrabarti, M. N. Garofalakis, R. Rastogi et al., Approximate query processing using wavelets, *International Conference on Very Large Data Bases*, Morgan Kaufmann Publishers Inc., 2000.
[20] E. J. Stollnitz, T. D. Derose and D. H. Salesin, *Wavelets for Computer Graphics: Theory and Applications*, 1996.
[21] F. Dubeau, S. Elmejdani and R. Ksantini, Non-uniform Haar wavelets, *Applied Mathematics & Computation*, vol.159, no.3, pp.675-693, 2004.
[22] A. Aboulnaga and S. Chaudhuri, Self-tuning histograms: Building histograms without looking at data, *ACM SIGMOD International Conference on Management of Data*, pp.181-192, 2001.
[23] L. Lim, M. Wang and J. S. Vitter, SASH: A self-adaptive histogram set for dynamically changing workloads, *International Conference on Very Large Data Bases*, pp.369-380, 2003.
[24] N. Bruno, S. Chaudhuri and L. Gravano, STHoles: A multidimensional workload-aware histogram, *Acm Sigmod Record*, vol.30, no.2, pp.211-222, 2001.
[25] U. Srivastava, P. J. Haas, V. Markl et al., ISOMER: Consistent histogram construction using query feedback, *International Conference on Data Engineering*, p.39, 2006.
[26] X. Lin, X. Zeng and X. Pu, CETLH: A new histogram-based cardinality estimate approach, *International Journal of Innovative Computing Information and Control*, vol.11, no.1, pp.123-135, 2015.
[27] R. Viswanathan, P. Jain, S. Laxman et al., A learning framework for self-tuning histograms, *Computer Science*, 2011.

[28] A. Khachatryan, E. Muller, C. Stier et al., Improving accuracy and robustness of self-tuning histograms by subspace clustering, *IEEE Trans. Knowledge & Data Engineering*, vol.27, no.9, p.1, 2015.
[29] J. A. Hartigan, Clustering algorithms, *Applied Statistics*, vol.23, no.6, pp.38-41, 1975.
[30] M. Mittal, R. K. Sharma and V. P. Singh, Modified single pass clustering with variable threshold approach, *International Journal of Innovative Computing Information and Control*, vol.11, no.1, pp.375-386, 2015.
[31] R. Stoica, J. J. Levandoski and P. A. Larson, Identifying hot and cold data in main-memory databases, *International Conference on Data Engineering*, pp.26-37, 2013.
[32] J. N. Darroch and D. Ratcliff, Generalized iterative scaling for log-linear models, *Annals of Mathematical Statistics*, vol.43, no.5, pp.1470-1480, 1972.
[33] D. Aha and P. Murphy, *UCI Repository of Machine Learning Databases*, http://archive.ics.uci.edu/ml/index.html, 2013.
[34] Transaction Processing Performance Council, *TPC Benchmark H Standard Specification Revision 2.17.0*, http://www.tpc.org/tpch/spec/tpch2.17.0.pdf, 2014.