Cardinality Estimation Based on Cluster Analysis

XIAONING ZENG1, XUDONG LIN1, CAIYAN PEI1, JING CAO1

1 College of Mathematics & Information Technology
Hebei Normal University of Science and Technology
Qinhuangdao, Hebei, 066004 P.R. China
E-mail: qhdzxn@163.com, peicaiyan78@163.com, caojxsf@163.com

2 Department of Information Engineering
Environmental Management College of China
Qinhuangdao, Hebei, 066004 P.R. China
E-mail: fydong_xt@126.com

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 (QFRs). We call this kind of cardinality estimation solutions the reactive solutions. In the existing reactive solutions, self-tuning histograms are constructed over the entire value ranges of the queried attributes and have large scales. The bucket number of a multi-dimensional self-tuning histogram increases exponentially with the dimension. That means the existing reactive solutions are stuck with the issue of “curse of dimension”. Simultaneously, to construct and maintain a multi-dimensional self-tuning histogram over the entire value ranges of the queried attributes, a large number of QFRs must be accumulated at a long time span, and the cumbersome operations have to be executed repeatedly to meet a space budget, which makes the existing reactive solutions unpredictable and time-consuming.

To address above issues, a new reactive solution is proposed in the paper. In the solution, 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 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 solution can be considered as a beneficial attempt to improve the cardinality estimation efficiency under high dimensions, and notably alleviate the issue of "curse of dimension". Furthermore, old QFRs can be replaced rapidly by the new QFRs and the data and workload changes can be timely reflected by micro self-tuning histograms. The process of meeting a space budget is eliminated completely, which makes the whole solution reliable and dexterous.

Keywords: Cardinality estimation; Cluster analysis; 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. For a predicate referring to multiple attributes, if the cardinality is estimated only based on 1-dimensional data summarization techniques such as 1-dimensional histograms or 1-dimensional wavelet transforms, the correlations among multiple attributes are ignored and the accuracy of cardinality estimation cannot be guaranteed. Therefore, 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 [1]. And then, the improved proactive solutions are proposed continuously [2][3][4][5][6][7]. The proactive solutions proposed in [8][9] are based on the multi-dimensional wavelet transforms [10][11]. But all of the existing proactive solutions are still in the experimental stage and no one is actually adopted in the mainstream databases. Two serious defi-
ciencies prevent these solutions being practical:

1. Lots of system resources are occupied by periodical data scans and the performance of routine queries are influenced seriously.

2. The solutions are stuck with the “curse of dimension”. As the dimension increases, the bucket number in a proactive histogram based solution or the wavelet coefficient number in a wavelet transform based solution increases exponentially.

The cardinality estimation solution in [12] 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 structure of a multi-dimensional self-tuning histogram can match the workload well and shows more details in the frequently queried areas. The succeeding reactive solution in [13] shows how to build low-dimensional self-tuning histograms from high-dimensional queries using the delta rule. The reactive solution in [14] improves the accuracy of a self-tuning histogram by subtilizing the granularity of QFRs. The information-theoretic principle of maximum entropy is introduced in [15] to construct a multi-dimensional self-tuning histogram which is consistent with all currently valid QFRs. [16] 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. [17] initializes a multi-dimensional self-tuning histogram based on subspace clustering. [18] tries to improve the efficiency of maintaining self-tuning histograms by constructing two level histograms.

Summing up the existing reactive solutions, periodical data scans are avoid and multi-dimensional self-tuning histograms can be constructed and maintained based on QFRs, but they are not yet practical due to the following common issues:

1. Although the periodical data scans are avoided, 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). To reflect the data distribution in the entire value range, a global self-tuning histogram must have a relatively large scale. As dimension increases, the bucket number of a global self-tuning histogram increases exponentially just as a proactive histogram. Although a global self-tuning histogram can be constructed during periods of light load, it is difficult even impractical to construct and maintain a global self-tuning histogram with large scale. Many system resources of database servers are occupied and the efficiencies of routine queries are influenced seriously.

2. To construct and maintain a global self-tuning histogram over the entire value range, 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 is widely adopted in different reactive solutions. But the limitation to bucket number leads to accuracy deterioration of cardinality estimation especially in the multi-dimensional cases. Furthermore, to meet a space budget, the process of reducing bucket number may be called more than once, and the complex and cumbersome operations included in the process have to be executed repeatedly, which makes reactive solutions unpredictable and time-consuming.

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 is executed, the Ward’s minimum variance method (Ward method for short) [19] 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:

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 $p$. If these predicates can be found, they will be very helpful to estimate the cardinality of $p$. The Ward method is exactly used to find these predicates according to its function of clustering simi-
The self-tuning histogram covering the entire value range of the queried attributes is abandoned. To estimate the cardinality of a new predicate, 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”.

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, which furthest guarantee the accuracy of cardinality estimation.

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 gives the notations. Section 3 describes the details of finding k nearest QFRs using the Ward method. The micro histogram and its construction process are analyzed in Section 4. Based on the micro histogram, the processes of cardinality estimation for different cases are given in Section 5. The new QFR update mechanism is explained in Section 6. The results of extensive experiments are demonstrated in Section 7. Section 8 discusses the related work, and Section 9 summarizes the paper and discusses future directions.

### 2. Notations

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

<table>
<thead>
<tr>
<th>Notations related to an attribute</th>
<th>meanings</th>
</tr>
</thead>
<tbody>
<tr>
<td>$min(x)$, $max(x)$</td>
<td>the minimum and the maximum of the data set $x$</td>
</tr>
<tr>
<td>$</td>
<td>x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Notations related to a predicate</th>
<th>meanings</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$ and its subscripted forms</td>
<td>predicate: a predicate $p$ has the form $(v_1 \leq a_1 \leq v_2), (v_1 \leq a_2 \leq v_2), \ldots, (v_1 \leq a_n \leq v_2)$ where $a_1, a_2, \ldots, a_n$ are the different attributes in one relation. $v_1$ and $v_2$ are two values within the value range of $a$, which satisfy $v_1 \leq v_2$ for $i = 1, 2, \ldots, m$. 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.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Notations related to a bucket</th>
<th>meanings</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d(a)$</td>
<td>the value range of $a$</td>
</tr>
<tr>
<td>$d(b, a)$</td>
<td>the value range of $a$ which is covered by $b$</td>
</tr>
<tr>
<td>$d(p, a)$</td>
<td>the value range of $a$ where $p$ is true</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Notations related to a histogram</th>
<th>meanings</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h$ and its subscripted forms</td>
<td>histogram</td>
</tr>
<tr>
<td>$b$ and its subscripted forms</td>
<td>the bucket of a histogram</td>
</tr>
<tr>
<td>$d(h)$</td>
<td>the value range of $b$: for a bucket $b$ of $h$ over the attributes $a_{i_1}, \ldots, a_{i_q}$ in the relation $r$, $d(b) = [\min(d(b, a_{i_1})), \max(d(b, a_{i_1}))] \ast \cdots \ast [\min(d(b, a_{i_q})), \max(d(b, a_{i_q}))]$</td>
</tr>
<tr>
<td>$f(b)$</td>
<td>the frequency of $b$, i.e., the number of tuples which fall in $b$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Notations related to a histogram</th>
<th>meanings</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d(h)$</td>
<td>the value range of $h$: a $k$-dimensional histogram $h$ over the attributes $a_{i_1}, \ldots, a_{i_q}$ in the relation $r$ is obtained by partitioning the value space $[\min(d(a_{i_1})), \max(d(a_{i_1}))] \ast \cdots \ast [\min(d(a_{i_q})), \max(d(a_{i_q}))]$ into one or more buckets and records the number of tuples falling in each buckets. $d(h) = [\min(d(a_{i_1})), \max(d(a_{i_1}))] \ast \cdots \ast [\min(d(a_{i_q})), \max(d(a_{i_q}))]$</td>
</tr>
<tr>
<td>$B(h)$</td>
<td>the bucket set composed of all buckets of $h$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Notations related to a predicate</th>
<th>meanings</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n(p)$</td>
<td>the real number of tuples which satisfy the predicate $p$</td>
</tr>
<tr>
<td>$exp(p)$</td>
<td>the estimated number of tuples which satisfy the predicate $p$</td>
</tr>
<tr>
<td>$exp(p)$</td>
<td>the estimated number of tuples which satisfy the predicate $p$</td>
</tr>
<tr>
<td>$d(p)$</td>
<td>the value range of $p$: for a $p = (v_1 \leq a_1 \leq v_2), (v_1 \leq a_2 \leq v_2), \ldots, (v_1 \leq a_n \leq v_2)$, $d(p) = \ldots$</td>
</tr>
</tbody>
</table>
\[ s(p) \quad \text{The dimension of } p \text{ for a } p = \left( v_1 \leq a_1 \leq v_2, v_1 \leq a_2 \leq v_2, \cdots, v_1 \leq a_n \leq v_2 \right), s(p) \text{ denotes the number of attributes } a_1, a_2, \cdots, a_n. \]

\[ q(f(p)) \quad \text{the QFR of } p: q(f(p)) = (p, n(p), m), \text{ where } m \text{ is the executing moment of } p. \]

### 3. Find k Nearest QFRs Using Ward Method

#### 3.1 Ward Method

Cluster analysis [20][21] 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). It is a common technique for multivariate statistical analysis, and used in many fields including machine learning, information retrieval, etc.

Hierarchical cluster analysis is an important kind of cluster analysis method. Its intuition is the following: each one of \( n \) samples is categorized into one class to compose \( n \) classes totally. And then, the distance between any two classes is calculated and the two classes with the minimum distance are merged. After the merger, the distances between any two classes are recalculated and the process is always repeated until all samples belong to one class.

As a hierarchical cluster analysis 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 \( G_1, G_2, \ldots, G_k \). \( X_i \) is the vector composed of the relevant variables of the \( i \) sample in \( G_i \), and \( n_i \) denotes the number of samples in \( G_i \). \( \bar{X_i} \) is the center of gravity of \( G_i \). The SSD of \( G_i \) can be expressed as:

\[ S_i = \sum_{t=1}^{n_i} (X_{it} - \bar{X_i})^T (X_{it} - \bar{X_i}) \]  

(1)

It reflects the degree of dispersion of the internal samples of \( G_i \).

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:

\[ D_{pq} = \sqrt{S_r - S_p - S_q} \]  

(2)

This distance is called the Ward distance in the paper.

#### 3.2 Ward Distance between Predicates

In CEWM, each QFR corresponds to an executed predicate and all QFRs are stored in a QFR warehouse. We define 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 each new predicate, its real cardinality is decided by the data distribution in its value range, which can be reflected by the QFRs of the executed predicates which have the similar value ranges with the new predicate. To find these executed predicates, we define the Ward distance between predicates based on their value ranges.

For a predicate \( p = (v_1 \leq a_1 \leq v_2, v_1 \leq a_2 \leq v_2, \cdots, v_1 \leq a_n \leq v_2) \), the vector composed of its location variables can be expressed as:

\[ X_p = (v_{11}, v_{12}, v_{21}, v_{22}, \cdots, v_{m1}, v_{m2})^T \]  

(3)

Supposing the class \( G_{p_1} \) and \( G_{p_2} \) contain the predicate \( p_1 \) and \( p_2 \) respectively. Simultaneously, the class \( G_{p_1,p_2} \) 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:

\[ D_{p_1,p_2} = \sqrt{S_{p_1,p_2} - S_{p_1} - S_{p_2}} \]

\[ = \left( \sum_{i=1}^{2} (X_{p_1} - \bar{X}_{p_1,p_2})^T (X_{p_1} - \bar{X}_{p_1,p_2}) \right)^{0.5} \]  

(4)

#### 3.3 Algorithm of Finding k Nearest QFRs

Given a new predicate \( p_{n+1} \) and the QFRs \( q(f(p_1)), \ldots, q(f(p_n)) \) corresponding to the executed predicates \( p_1, \ldots, p_n \), finding \( k \) \((1 \leq k \leq n)\) nearest QFRs with \( p_{n+1} \) from \( q(f(p_1)), \ldots, q(f(p_n)) \) is to find \( k \) executed predicates from \( p_1, \ldots, p_n \) which have the 1st shortest Ward distance to the \( k \)th shortest Ward distance with \( p_{n+1} \). The algorithm of finding \( k \) nearest QFRs is composed of the following four steps:

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

Step 2: Merge \( G_{pn+1} \) with every one of \( G_{p_1}, \ldots, G_{p_n} \) respectively and calculate the Ward distances
The pseudo-codes of above steps are shown as Algorithm 1:

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

$$\text{knnfr}(p_{ni}, p[1\cdots n])$$

1. $G_{\text{set}}\leftarrow p_{ni}$
2. for each $i \in [1\cdots n]$ do
3. \hspace{1em} $G_{ij} \leftarrow p[i]$
4. \hspace{1em} $\text{wd}[i] \leftarrow \text{calWardDist}(G_{\text{set}}, G_{ij})$ \hspace{0.5em} //calculating Ward distances
5. \hspace{1em} $\text{loc} \leftarrow \text{sort}(\text{wd}[1], \cdots, \text{wd}[n])$ \hspace{0.5em} //sorting Ward distances
6. \hspace{1em} $k \leftarrow \text{configK}(p_{ni}, p[1\cdots n], \text{loc}[1\cdots n])$ \hspace{0.5em} //configuring $k$, elaborated in Algorithm 2
7. \hspace{1em} return $\text{loc}[1\cdots k]$

It is necessary to emphasize that the purpose of introducing the Ward method in Algorithm 1 is not merging $G_{\text{set}}$, $G_{p1}$, $\cdots$, $G_{pn}$ into one class to realize the intact cluster analysis for the query predicates $p_{ni}$ and $p_{1}$, $\cdots$, $p_{n}$, but calculating and sorting the Ward distance between $p_{ni}$ and $p_{1}$, $\cdots$, $p_{n}$.

For Algorithm 1, the pseudo-codes in Line 2 to Line 4 calculate all Ward distances with the time complexity of $O(n*2s(p_{ni}))$ because one predicate corresponds to two coordinates in each dimension. The pseudo-code in Line 5 sorts the Ward distances with the time complexity of $O(n*\log(n))$. From Section 3.4, we can know the time complexity of the pseudo-code in Line 6 is $O(UL^2*4s(p_{ni}))$ where $UL$ is the upper limit of $k$ value and will be explained in Section 3.4. In general, $s(p_{ni})$ and $UL$ are less than 10. Therefore, the time complexity of the whole Algorithm 1 is about $O(m*n)$ where $m$ is a number far less than $n$. The experimental results can fully testify the conclusion. For a new predicate, it spends few than 100ms finding its $k$ nearest QFRs from the QFR warehouse storing several hundreds of QFRs.

For example, a 2-dimensional new predicate $p_{ni}$ and ten executed predicates $p_{1}$, $\cdots$, $p_{10}$ are shown in Figure 1. The new predicate is shown as a grey rectangle and the rectangles with dashed borders denote the executed predicates. Configure $k=3$, the executed predicates corresponding to the $k$ nearest QFRs with $p_{ni}$ are shown as the rectangles with bold solid borders. They can be efficiently obtained by Algorithm 1.

![Figure 1. A new predicate and ten executed predicates](image)

### 3.4 The Choice of $k$ Value

Using Algorithm 1, the $k$ nearest QFRs with a new predicate can be found. But $k$ value must be chosen carefully. The purpose of finding $k$ nearest QFRs is to construct a micro histogram. And the size of a micro histogram is highly related to the $k$ value. To choose an appropriate $k$ value, the distribution feature of query workload should be analyzed firstly. 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 [22][23]. Figure 2 shows a skewed query workload example in a 2-dimensional space. Each predicate is denoted by a rectangle. From the figure, we can observe that the area surrounded by a rectangle with bold solid borders is more frequently queried than the other
area.

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, i.e., the value range union of the chosen predicates with the minimum number can fully cover the value range of the new predicate. For example, a new predicate \( p_{n1} \) locates in a frequently queried area and we can find \( d(p_{n1}) \subseteq \bigcup_{i=1}^{100} d(p_i) \). Assuming that \( D_{p_{n1}p_1} < \cdots < D_{p_{n1}p_{n0}} \), if \( d(p_{n1}) \not\subseteq \bigcup_{i=1}^{1} d(p_i) \) but \( d(p_{n1}) \subseteq \bigcup_{i=1}^{5} d(p_i) \), we can configure \( k=5 \).

For the new predicates locating in the infrequently queried areas, it is possible that no enough executed predicates whose value range union can fully cover the value range of the new predicate. But in CEWM, the cardinality estimation accuracy is much more decided by the predicates locating in the frequently queried areas, so 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 \( p_{i} \), the \( k \) value can be chosen as follow:
\[
k = \min(C_p, UL) \tag{5}
\]

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 \( C_p \leq UL \).

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 [15] will be executed to calculate \( k \) value. By drilling holes continuously inside \( d(p_{n1}) \) according to the executed predicates, either we can find the executed predicates whose value range union can fully cover \( d(p_{n1}) \), or we can confirm there are no enough predicates whose value range union can fully cover \( d(p_{n1}) \). 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 \( if((d_{uca} \subseteq d(p_{n1})) \)
11 \( ucas \leftarrow ucas - uca \)
```

---
else if \((d_{\text{el}} \cap d_{\text{loc}[i]} \neq \emptyset)\)

\[
\text{tempUcas} \leftarrow \text{tempUcas} \cup \text{holeDrilling}(p[\text{loc}[i]], \text{ucc})
\]

\[
\text{ucas} \leftarrow \text{ucas} \cup \text{tempUcas}
\]

return \(k\)

For Algorithm 2, the time complexity is \(O(UL^*|uccas|_{\text{max}}^*2s(p_{n1}))\). \(uccas\) is different in each outer loop beginning from Line 3, and \(|uccas|_{\text{max}}\) denotes the maximum of all \(|uccas|\) in the UL-1 outer loops. It is difficult to deduce the precise \(|uccas|_{\text{sum}}\) because the number of uncovered areas generated by each drilling hole operation can be different. From experiments, we know the drilling hole operations based on \(x\) predicates can generate 1.5\(x\) to 2.0\(x\) uncovered areas. Therefore, the time complexity of Algorithm 2 can be estimated as \(O(UL^*2^*4s(p_{n1}))\).

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 operation according to 3 executed predicates, no any uncovered areas can be found. Therefore, the \(k\) value can be configured as 3.

![Figure 3. Process of configuring k value](image)

### 4. Construct Micro Histogram

According to the \(k\) nearest QFRs with a new predicate \(p\), the most direct way to estimate the cardinality of \(p\) is to average the real cardinalities of all \(k\) corresponding executed predicates and the entire value range of \(p\) is considered as an area with a uniform data distribution. But in an actual database, it is common that different areas in the value range of \(p\) have the non-uniform data distribution. So we choose to construct a micro histogram for \(p\) based on its \(k\) nearest QFRs to estimate its cardinality. The different frequencies of the buckets in the micro histogram can reflect the non-uniform data distribution in the value range of \(p\).

#### 4.1 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. In CEWM, to estimate the cardinality of a new predicate, a micro histogram will be constructed and the covered area of the micro histogram is determined by the new predicate.

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 a micro histogram only contains very few buckets and the number of QFRs participating in the construction of a micro histogram is very limited; 3) due to the efficient performance of construction, an micro histogram can be constructed for each new predicate, and the changes of the underlying data can be embedded into a micro histogram in a more timely manner.

#### 4.2 Construct Micro Histogram

In CEWM, we do not propose the new histogram structure, but adopt the structure in ISOMER [15] to construct a micro histogram. This structure is improved from STHoles [14] by replacing the
“artificial feedback” with the actual feedback that naturally occurs during query execution. And this structure is also superior to the histogram structures in MHiSt [2], STGrid [12] and SASH [13].

For a multi-dimensional micro histogram \( h \) constructed over a certain local of the value ranges of the attributes \( a_{r1}, \ldots, a_{rk} \) in the relation \( r \), each bucket \( b \in B(h) \) covers a hyper rectangle with two constant boundaries in each dimension. But the value range of \( b_i \) is not the entire value range of the hyper rectangle. Inside each hyper rectangle, there may be some mutually disjoint sub hyper rectangles which are covered by the other buckets \( b_{j1}, \ldots, b_{jk} \). We say \( b_{j1}, \ldots, b_{jk} \) 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 follow:

\[
d(b_i) = d(R_{b_i}) - \bigcup_{x=1}^{k} d(b_{ix})
\]

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(a) and Figure 4(b) respectively.

The histogram construction algorithm of CEWM is a simpler version of the algorithm in ISOMER:

![Figure 4. A micro histogram and the corresponding bucket tree](image)

MER. Specifically, the construction of a multi-dimensional micro histogram in CEWM includes two steps: firstly, the drilling hole operation is executed to get the value range of each bucket; secondly, the frequency of each bucket is calculated based on the information-theoretic principle of maximum entropy [24][25]. The time-consuming merging operation and the corresponding histogram recalculations in ISOMER are avoided.

For each new predicate \( p \) and its \( k \) nearest QFRs \( qfr(p_1), \ldots, qfr(p_k) \), a micro histogram \( h \) is initialized with one bucket covering the value range \( d(p_i) \). 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 [26] will be loaded as an approximation of the information-theoretic principle of maximum entropy to calculate the frequencies of all buckets in \( h \). And the calculated frequencies have the maximum entropy value for the probability distribution defined in [15] and satisfy all \( k \) nearest QFRs \( qfr(p_1), \ldots, qfr(p_k) \) with \( p \).

The pseudo-codes of constructing a micro histogram are shown as Algorithm 3:

```
Algorithm 3: constructing the micro histogram \( h \) for the new predicate \( p_1 \)

\( \text{constructHist}(p_0, p_1, \ldots, p_k, \text{loc}[\ldots]) \)
1 \( T_{\text{init}} \leftarrow T_{\text{init}} \cup b_0 \) //The bucket tree \( T_{\text{init}} \) is initialized with the bucket \( b_0 \) covering the whole \( d(p_0) \).
2 \( \text{for each } i \in \text{loc}[\ldots] \) do
3 \( \text{if } (\exists \text{bucket } b^* \text{ satisfying } d(b^*) = d(p_i)) \)
4 \( T_{\text{new}} \leftarrow b^* \)
5 \( \text{for each } (b \in T_{\text{new}}) \times (d_i \cap d_{ij} = \emptyset) \) do
6 \( T_{\text{new}} \leftarrow T_{\text{new}} \cup \text{holeDrilling }(p_i, b) \)
7 \( T_{\text{init}} \leftarrow T_{\text{init}} \cup b \)
8 \( T_{\text{init}} \leftarrow T_{\text{init}} \cup T_{\text{new}} \)
9 \( h \leftarrow (T_{\text{init}}, \text{m}(\text{loc}[\ldots]), \ldots, \text{m}(\text{loc}[\ldots])) \) //executing the IS algorithm
```

For Algorithm 3, the pseudo-codes in Line 1 to Line 8 generates all buckets in the micro histogram.
Definition 1. Cardinality Estimation. Before executing a predicate \( p \), the estimation about the tuples satisfying \( p \) is called the cardinality estimation of \( p \).

Cardinality estimation is an important problem in query optimization. Assuming the micro histogram \( h \) is constructed to estimate the cardinality of the new predicate \( p \). For a bucket \( b \in B(h) \) satisfying \( d(p) \cap d(b) \neq \emptyset \), the number of tuples which fall in \( b \) and satisfy \( p \) can be estimated according to the uniformity assumption:

\[
est_b(p) = f(b) \cdot |d(b) \cap d(p)|/|d(b)|
\]  

(7)

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

Case 1: if there exist the bucket set \( B'(h) \subseteq B(h) \) satisfying \( d(p) \subseteq \bigcup_{b \in B'(h)} d(b) \), and for each \( b \in B'(h) \), \( d(p) \cap d(b) \neq \emptyset \), the cardinality of \( p \) is:

\[
est(p) = \sum_{b \in B'(h)} \nest_b(p)
\]  

(8)

Case 2: if no bucket set \( B'(h) \subseteq B(h) \) satisfying \( d(p) \subseteq \bigcup_{b \in B'(h)} d(b) \) can be found, but there exist a bucket set \( B''(h) \subseteq B(h) \) satisfying \( d(p) \cap d(b) \neq \emptyset \) for each \( b \in B''(h) \), the cardinality of \( p \) is:

\[
est(p) = \sum_{b \in B''(h)} \nest_b(p) \cdot |d(p)|/\sum_{b \in B''(h)} |d(b) \cap d(p)|
\]  

(9)

Case 3: if no bucket \( b \in B(h) \) satisfies \( d(p) \cap d(b) \neq \emptyset \), the cardinality of \( p \) is:

\[
est(p) = \sum_{b \in B(h)} f(b) \cdot |d(p)|/\sum_{b \in B(h)} |d(b)|
\]  

(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.

For a new predicate \( p \) in Case 1, the data distribution of its entire value range can be approximated by the micro histogram \( h \), so its cardinality can be estimated using (8) and the error of cardinality estimation only originates from the uniformity assumption.

For a new predicate \( p \) in Case 2, only the data distribution of parts of its value range which are covered by the buckets in \( B''(h) \) can be approximated by the micro histogram \( h \), so its cardinality can be estimated using (9) and the extra error of cardinality estimation may be introduced by estimating the entire cardinality in the value range \( d(p) \) with the local cardinality in the value range \( \bigcup_{b \in B''(h)} (d(b) \cap d(p)) \).

For the new predicate \( p \) in Case 3, no information about the data distribution in its value range can be provided by the micro histogram \( h \). Therefore, its cardinality has to be estimated using (10) by aver-
aging the frequencies of all buckets of $h$, and the remarkable error of cardinality estimation may be caused by the possible different data distributions between $d(p)$ and $d(h)$.

It is mentioned that an actual query workload shows a certain skewed distribution feature in general. For a query optimizer, both the holistic accuracy of cardinality estimation and the effect of choosing optimal query plan are much more decided by the predicates locating in the frequently queried areas. Due to the high degree of overlap of the predicates in the frequently queried areas, it is common that the cardinalities of such predicates can be estimated using (8) or (9). Therefore, the holistic accuracy of cardinality estimation can be fully guaranteed in CEWM.

6. QFR UPDATE MECHANISM

Before elaborating the QFR update mechanism in CEWM, we analyze which QFRs should be replaced. Firstly, based on the Ward method, the $k$ nearest QFRs can be found for a new predicate $p$. And the $k$ corresponding executed predicates locates in the same or adjacent area with $d(p)$. That means $p$ and the $k$ corresponding executed predicates can reflect the data distribution of the same or adjacent area. So it is reasonable to choose a replaced QFR from the $k$ nearest QFRs with $p$. Secondly, it is rational to consider that for two predicates in a same or adjacent area, the newer one can reflect the current data distribution in the area more accurately than the older one.

Based on above analyses, 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

![Figure 6. Cardinality estimations in different cases](image)

![Figure 7. QFR updates without query workload changes](image)
warehouse are shown in Figure 7(b) to Figure 7(d) respectively (Some medium statuses are omitted).

For $p_{a1}$, shown as a grey rectangle in Figure 7(b), $k=3$ and the $k$ nearest QFRs obtained by the Ward method are shown as rectangles with bold borders. We assume that the oldest one is shown as a rectangle with dashed bold borders, and it will be replaced by $qfr(p_{a1})$ after $p_{a1}$ is executed.

For $p_{a2}$ to $p_{a10}$, the corresponding QFRs can be updated in the same manner as $p_{a1}$. $qfr(p_1)$ to $qfr(p_{10})$ can be replaced by $qfr(p_{a1})$ to $qfr(p_{a10})$ gradually, and the final status of the QFR warehouse is shown in Figure 7(d). The possible underlying data changes can be timely reflected by the new QFRs.

We also assume the other ten successively executed new predicates $p_{a11}$ to $p_{a20}$ have the different distribution feature with $p_1$ to $p_{10}$. Based on the QFR update mechanism in CEWM, the changes of the QFR warehouse are shown in Figure 8(b) to Figure 8(d) respectively (Some medium statuses are omitted).

In Figure 8(b), the new predicate $p_{a11}$ which is shown as a grey rectangle locates in an area which has not been queried ever. So we can configure $k=UL$, and the $k$ nearest QFRs with $p_{a11}$ can be obtained by the Ward method. After cardinality estimation, the oldest one will be replaced by $qfr(p_{a11})$.

With the same manner, $qfr(p_1)$ to $qfr(p_{10})$ can be replaced by $qfr(p_{a11})$ to $qfr(p_{a20})$ gradually. At this process, the frequently query area moved from $A_1$ to $A_2$ which is shown in Figure 8(d). Therefore, we say the QFR update mechanism in CEWM can make a micro histogram matches not only the changes of underlying data but also the changes of query workload distribution.

7. 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.

7.1 Experimental settings

7.1.1 Data sets

To test CEWM comprehensively, a real data set [27] and the TPC-H benchmark [28] 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.

7.1.2 Query workloads

In our experiments, two query workload models $qw_1$ and $qw_2$ which follow the Zipfian distribution [29] and the Gaussian distribution [30] 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.
\( qw_1 \): \( qw_1 \) denotes the query workload model which follows the 2-dimensional Zipfian distribution with the skew parameter \( z=1 \). The whole domains of the queried attributes are divided into 5 mutually disjoint parts. The satisfactery of the 2-dimensional Zipfian distribution includes two aspects: (1) the sizes of 5 mutually disjoint parts follow the 2-dimensional Zipfian distribution; (2) the numbers of predicates within 5 mutually disjoint parts follow the reverse 2-dimensional Zipfian distribution.

\( qw_2 \): \( qw_2 \) denotes the query workload model follows the superposition of three 2-dimensional Gaussian distributions, and 3 median pairs selected at random from the whole domain of the queried attributes. Around each median pair, the predicates are generated following a 2-dimensional Gaussian distribution with the same standard deviation \( d \). The parameter \( d \) is configured as 5 percent of the width of the domains of the queried attributes.

### 7.1.3 Metrics

Firstly, we can define \( re(p) \), the relative error of a predicate \( p \) using (11):

\[
re(p) = \frac{abs(n(p) - est(p))}{n(p)}
\]

(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:

\[
rar(ce) = \frac{cn_s(ce)}{tn(ce)}
\]

(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.

### 7.1.4 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.

Static experiments and dynamic experiments will be carried out based on each combination of data set and query workload. Static experiments mean the underlying data and the distribution features of query workloads remain unchanged when the experiments are carried out. And dynamic experiments allow the changes of the underlying data and the distribution features of query workloads during the experiments.

### 7.2 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, and the relative accuracy rate and the overall execution time will be recorded for each solution.

The results of the static experiments based on \( ds_i \) and \( qw_1 \) are shown in Figure 9. From Figure 9(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. Furthermore, in Figure 9(a), we can observe obvious fluctuations from the relative accuracy rate curves of ISO2 and ISO3. By comparing the histograms constructed by ISOMER and the actual distributions of the underlying data, we find that sometimes, a histogram cannot match the underlying data well. And the matching is much related to the larger number of equations participating in the calculation. In ISOMER, several hundreds of QFRs are needed to calculate a global histogram together. In this case, the IS algorithm adopted in ISOMER shows some instability and cannot always provide accurate results. For CEWM, the adopted micro histograms only contain few buckets and the IS algo-
Algorithm can always provide the relatively accurate results.

From Figure 9(b), we can also see the superiority of CEWM in efficiency. The time complexity analyses of Algorithm 1 and Algorithm 3 have indicated the high efficiency of CEWM in theory. And in the actual efficiency experiments, CEWM can finish the cardinality estimations of each 100 predicates within 10 seconds in general.

But 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 than the ISO2 solution, the efficiencies drop 70 percent averagely.

Figure 9II shows the results of the static experiments based on $d_{s1}$ 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 with the ones in Figure 9I. CEWM is still accurate, stable and efficient. And the fluctuation of ISOMER is still obvious.

Figure 9III shows the results of the static experiments based on $d_{s2}$ 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 shows much more superiority in efficiency. It 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 $d_{s2}$ and $qw_2$ are shown in Figure 9IV. We can see more fluctuations from the relative accuracy rate curves of CEWM. But in fact, the difference between the minimum and the maximum of the relative accuracy rates of CEWM are only 3.5 percent. The tiny difference of skewness between $d_{s1}$ and $d_{s2}$ is the main reason leading to the slight deterioration of CEWM in accuracy and stability over $d_{s2}$.
The averages in different static experiments are summarized in Table 2. CEWM shows excellent performance in static experiments. Compared with ISOMER, CEWM can finish cardinality estimations more accurately in much shorter time. But in an actual database, the underlying data and the query workloads are always changing. Therefore, only the static experiments are not enough, and the practicability of an solution must be further tested based on the dynamic experiments.

<table>
<thead>
<tr>
<th>Solutions</th>
<th>averages of RAR(%)/Time(s)</th>
<th>$d_3$ and $qw_1$</th>
<th>$d_3$ and $qw_2$</th>
<th>$d_5$ and $qw_1$</th>
<th>$d_5$ and $qw_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEWM</td>
<td>85.8/5.2</td>
<td>83.2/5.5</td>
<td>96.1/4.1</td>
<td>93.2/2.9</td>
<td></td>
</tr>
<tr>
<td>ISO2</td>
<td>55.4/122.4</td>
<td>51.0/121.3</td>
<td>88.4/89.1</td>
<td>88.3/89.0</td>
<td></td>
</tr>
<tr>
<td>ISO3</td>
<td>63.1/201.1</td>
<td>58.2/192.5</td>
<td>92.9/137.4</td>
<td>90.9/131.3</td>
<td></td>
</tr>
</tbody>
</table>

### 7.3 Dynamic experiments

The preparation stage and the formal stage of each dynamic experiment is similar with 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 $d_3$, 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 $d_3$ can be updated. For the data set $d_5$, 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 $d_5$ can be updated.

As the underlying data are updated by the refresh functions, the query workloads are also changed. For the data set $d_3$, 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 $d_5$, 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 $d_3$ and $qw_2$ are shown in Figure 10I. Compared with the results of the corresponding static experiments in Figure 8, 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 is paid out, the ISO3 solution can only provide accurate cardinality estimations for few 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 increase from 5.9 percent to 11.3 percent, which show that the changing underlying data and query workloads bring larger fluctuations. But compared with the fluctuations of 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 10I(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. For the cardinality estimation of a predicate, the calculation of the corresponding micro histogram only needs few similar QFRs with the predicate, which improves the accuracy of the micro histogram calculation and the corresponding cardinality estimation. Simultaneously, no matter the underlying data or the query workload change, each predicate
can bring the new information about the changes continuously and timely. On the contrary, ISOMER can only update QFRs periodically and the changes of underlying data during two updates cannot be reflected to the histogram timely. And when ISOMER discards outdated QFRs, it considers the “age” of a QFR much more, but the similarities among QFRs cannot play their roles, which deteriorate the accuracy of histogram calculation and the corresponding cardinality estimation.

Figure 10 shows the results of the dynamic experiments based on $d_{s1}$ and $qw_1$. Using a query workload following the 2-dimensional Gaussian distribution, the changing tendencies of the relative accuracy rate and the execution time are similar with the ones in Figure 13.

Figure 10II shows the results of the dynamic experiments based on $d_{s1}$ and $qw_2$. Using a query workload following the 2-dimensional Gaussian distribution, the changing tendencies of the relative accuracy rate and the execution time are similar with the ones in Figure 13.

Figure 10III shows the results of the dynamic experiments based on $d_{s2}$ and $qw_1$. Apparently, the relative accuracy rates of all solutions in the figure are better than the ones in the dynamic experiments over the skewed underlying data. Most relative accuracy rates of CEWM are higher than 90 percent. Although the difference between the relative accuracy rates of CEWM and the ISO3 solution can reduce to only 6 percent for the 400th to the 500th predicates, 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 10III(b), we can still observe the superiority of CEWM in efficiency.

The results of the dynamic experiments based on $d_{s2}$ and $qw_2$ shown in Figure 10IV show more fluctuations than the ones shown in Figure 10III.

The averages in different dynamic experiments are summarized in Table 3. CEWM can adapt to the changes of the underlying data and the query workloads much better than ISOMER: firstly, CEWM shows stability and robustness in all dynamic experiments. Over the changing data and query workloads, CEWM can provide accurate cardinality estimations for more than 70 percent validation predicates in the worst case. Simultaneously, the changes of underlying data and query workloads have little effect on the efficiency of CEWM, and the cardinality estimation for each validation predicate can be finished within several 10 milliseconds.
Table 3. Averages in different dynamic experiments

<table>
<thead>
<tr>
<th>Solutions</th>
<th>Average of RAR(%)/Time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$d_s$ and $qw_1$</td>
</tr>
<tr>
<td>CEWM</td>
<td>77.3/6.6</td>
</tr>
<tr>
<td>ISO2</td>
<td>39.3/123.5</td>
</tr>
<tr>
<td>ISO3</td>
<td>40.5/199.6</td>
</tr>
</tbody>
</table>

7.4 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 $d_s$ and $qw_1$ are shown in Figure 11, 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, and the cardinalities of some predicates locating in the frequently queried areas cannot be estimated accurately because no enough similar QFRs of the executed predicates can be found. But when the initial capacity of the QFR warehouse increases to 300, the frequently queried areas can be fully covered by the QFRs of the executed predicates and the accuracy of cardinality estimation can be guaranteed. In this case, it is unnecessary to increase the QFR density continuously because it is not helpful to improve the accuracy of CEWM prominently.

From Figure 11(b), we know as the increase of the initial capacity of the QFR warehouse, the cardinality estimation time only increases a little and the efficiency of CEWM cannot be influenced by the initial capacity of the QFR warehouse remarkably.

Figure 11II shows the experimental results of the initial capacity of the QFR warehouse based on $d_s$ and $qw_2$. No remarkable difference can be observed from the results in Figure 11.

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 $d_s$ and $qw_1$ are shown in Figure 12I. And Figure 12II shows the results of the dynamic experiments based on $d_s$ and $qw_2$. In Figure 12, the upper limit of $k$ value (denoted by $k$) is configured as 10, 20, 30, 40 and 50 respectively. We know the upper limit of $k$ value play its role only when the Condition 2 in Section 3.4 is satisfied. Therefore, the cardinality estimations of the predicates locating in the frequently queried areas are almost not related to the upper limit of $k$ value. No matter which upper limit of $k$ value is adopted, the results of cardinality estimations of these predicate are same. Therefore, we cannot see obvious difference from the relative accuracy
The first solution which applies the 1-dimensional histogram in cardinality estimation is proposed in [31]. And then, the improved solutions [1][2][3][4][5] 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 [35][36], DB2[37][38] and SQL Server[39] respectively. To keep a 1-dimensional histogram consistent with changing data and query workload distribution, the underlying data is scanned periodically in an actual database.

But for a predicate referring to multiple attributes, the cardinality estimation based on a 1-dimensional histogram relies on the independence assumption [40][41] and the correlations among multiple attributes are ignored, which leads to serious cardinality estimation errors inevitably. Multi-dimensional histograms have more practical values than 1-dimensional histograms for the cardinality estimation because few actual data satisfy the attribute value independence assumption. Although multi-dimensional histograms have not been adopted in the mainstream relational databases, the experimental multi-dimensional histograms have been researched for many years [1][2][3][4][5]. But in the existing solutions, the bucket number in a multi-dimensional histogram will increase exponentially with dimensionality and it is very time-consuming to maintain a multi-dimensional histogram according to the data changes. This is the main reason that no multi-dimensional histogram is adopted in the mainstream databases until now.

The wavelet transform is a useful mathematical tool for hierarchically decomposing datasets in ways that are both efficient and theoretically sound [10][11]. Some cardinality estimation solutions are based on wavelet transforms [8][9]. Compared with histograms, wavelet transforms are different in the
following aspects: (1) wavelet transforms are better than histograms in accurately modeling datasets with a larger number of discontinuities [42]; (2) for a value range, the partitions of wavelet transform are constrained to the dyadic collections over the value range, while histograms can form any arbitrary partitions of the value range.

[43] begins to use QFRs to help estimate cardinality, but no histogram is constructed in [43]. The first self-histogram based on QFRs appears in [12]. The solutions in [12] and [13] adopt heuristics mechanism to update self-tuning histograms, which leads to inconsistency between a histogram and the corresponding QFRs. The solution in [14] improves the accuracy of a self-tuning histogram by subutilizing the granularity of QFRs, but the extremely detailed feedback information requirements in the solution leads to expensive time cost. The self-tuning histogram structure in [14] is also used in [15], but the histogram in [15] can be consistent with all currently valid QFRs by introducing the information-theoretic principle of maximum entropy. [44] proposes an alternative formulation for consistency to improve the performance of the maximum entropy based solution. [45] leverages all available query feedback information based on the information-theoretic principle of maximum entropy and is scalable to multiple dimensions and large number of QFRs. [16] proposes a simple learning-theoretic formalization of self-tuning histograms. A self-tuning histogram is learned based on QFRs to alleviate the issue of “curse of dimension” and improve the efficiency of the solution. Simultaneously, 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. 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 that our solution is satisfactory in the accuracy and the efficiency of cardinality estimation.

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

9. 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. Simultaneously, 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. 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 that our solution is satisfactory in the accuracy and the efficiency of cardinality estimation.

ACKNOWLEDGMENT

This work is supported in part by the research funds from the science and technology plan of Hebei Province under Grant No. 16210348, and the humanities and social sciences major research project of Hebei Provincial Department of Education under Grant No. ZD201710. We would like to thank the anonymous reviewers for their constructive suggestions.

REFERENCES

using wavelets,” in Proceedings of the ACM SIGMOD International Conference on Management of
wavelets,” In Proceedings of the International Conference on Very Large Data Bases, pp. 111–122,
Cairo, Egypt, September, 2000 (42).
ence (2011).
Subspace Clustering," IEEE Transactions on Knowledge & Data Engineering, 27(9), pp. 1-1
[19] Murtagh F, Legendre P. Ward's Hierarchical Clustering Method: Clustering Criterion and Agglom-
[22] Y. E. Ioanidis, and V. Poosala, “Balancing Histogram Optimality and Practicality for Query Re-
sult Size Estimate,” In SIGMOD, pp.233-244, 1995.
[23] J. J. Levandoski, P.A. Larson, and R. Stoica, Identifying hot and cold data in main-memory data-
2003.
[25] V. Markl, P. Haas, M. Kutsch, N. Megiddo, U. Srivastava and T. Tran, “Consistent selectivity es-
[27] D. Aha, and P. Murphy, “UCI repository of machine learning databases,”
[29] G.K. Zipf, “Human Behavior and the Principle of Least Effort,” Addison-Wesley, Reading, MA,
1949.
Electrical Engineering and Computer Science, Case Western Reserve University, 1980.


