Reduced Quotient Cube: Maximize Query Answering Capacity in OLAP

QUANKUN WANG\textsuperscript{1}, JINGUO YOU\textsuperscript{1,2}, BENYUAN ZOU\textsuperscript{1}, YU CHEN\textsuperscript{1}, XINGRUI HUANG\textsuperscript{1}, AND LIANYIN JIA\textsuperscript{1,2}
\textsuperscript{1}Faculty of Information Engineering and Automation, Kunming University of Science and Technology, Kunming, Yunnan 650500, China
\textsuperscript{2}Yunnan Key Laboratory of Artificial Intelligence, Kunming University of Science and Technology, Kunming, Yunnan 650500, China
Corresponding author: Jinguo You (jgyou@kust.edu.cn)

This work was supported in part by the Natural Science Foundation of China under Grant 62062046 and Grant 61462050.

ABSTRACT The data cube is a critical tool for accelerating online analysis in big data. Due to its exponential space overhead, the quotient cube, as the main data cube compression approach, was proposed to significantly reduce the number of data cells if they are aggregated over the same base tuple set, i.e. they are cover equivalent to form an equivalence class. Nevertheless, it still poses challenges to efficiently analyze massive data due to high storage space consumption. This paper proposes the reduced quotient cube (RQC) based on the following observation. (i) there are equivalence classes of various sizes in a quotient cube; (ii) the small equivalence classes usually dominate; (iii) the big equivalence classes are more capable of query answering since they can induce more data cells. Unlike the quotient cube, which preserves all the equivalence classes of equal priority, the reduced quotient cube preferentially does those with larger query answering capacity and smaller space occupied capacity. Further, we design its efficient constructing and querying algorithms. The extensive experimental results show that compared with the quotient cube, the reduced quotient cube space is only 11.3\%, while the maximum query capacity is 95.9\%. The query time of the reduced quotient cube is reduced by 51.24\% on average compared to the quotient cube.

INDEX TERMS Data cube, equivalence class, OLAP, quotient cube, storage optimization.

I. INTRODUCTION
In recent years, with the continuous development of big data and data warehouses \cite{1}–\cite{3}, it is still a big challenge to analyze and process massive data. OLAP \cite{4}–\cite{7} can analyze the data along the dimension hierarchy based on the historical data of the data warehouse so that a large number of detailed data can be described in a more concise and general way, which is convenient for users to obtain the overall view. In a decision support system, the data is usually organized into data cubes \cite{8}. The core operation of the data cube is to aggregate the attributes of multiple dimensions of the data \cite{9}–\cite{12}. By pre-calculating the data cube, the data query time can be significantly reduced. However, the complete materialization of the data cube will lead to a dramatic increase in its storage scale. Because the data cube’s storage space complexity is exponential, the computation amount of the data cube will increase multiple times as the number of dimensions increases in practical data processing applications. Consequently, most related researches have been devoted to data cube compression.

The quotient cube \cite{13} is an essential method for compressing data cubes. Its intrinsic is that in the quotient cube lattice, there is a cover relation between different data cells, and the set of data cells formed by the cover relation is called the equivalence class. Since the data cell measures of the same class are equal, only the upper bound of the equivalence class needs to be saved, and the queries of other data cells can be obtained from the equivalence class’s cover relation, allowing the equivalence class to retain semantics while compressing data.

Lakshmanan, Pei, and Zhao further proposed a tree-like concise structure called QC-Tree \cite{14} to preserve the set of closed cells. When the equivalence class preserves only the upper bound (i.e., closed cells), it is called a closed cube \cite{15}. Condensed Cube \cite{16} and Dwarf Cube \cite{17} compressed by finding the same prefix or suffix between data cells are variants of the quotient cube. In addition to the quotient cube,
there are a number of other related efforts used to compress
data or optimize queries. [18] proposes to characterize the
parts of a data cube to be materialized with the help of
the FD’s (Functional Dependency) present in the underlying
data to reduce redundancy and optimize queries. Based on
the Frag-Shells method, an improved CFSC (Closed Frag-
Shells Cube) method is proposed in [19], and a query index
table of closed segments is constructed by using bit map
index to reduce the storage space occupied by the result
set and increase query efficiency. As it is rather costly to
support OLAP on big data, and the methods that compute
exact answers cannot meet the high-performance require-
ment, AOP (Approximate Query Processing) is proposed
to alleviate this problem. [20] discusses the problems and
challenges faced by AOP and surveys the existing methods
to solve these problems.

Although the preceding works have resulted in some
improvements in query efficiency and storage efficiency,
it still consumes a large amount of storage space when
processing massive data. This is due to the fact that the
traditional data cube algorithm is not designed to take into
account the user’s queries. Each equivalence class is assumed
the same responsiveness to the queries in the quotient cube,
i.e., equivalence classes with more data cells are wrongly
identified the same ability of query answer as those with
fewer data cells. In addition, because the data cube is sparse,
some equivalence classes have only a small number of data
cells, and such equivalence classes occupy a large amount of
space but have limited ability to answer queries. Therefore,
when the storage space is limited, it is valuable to retain the
equivalence class with a larger cover capacity.

Fig. 1 shows a four-dimensional quotient cube. The set
of data cells inside a circle with a label is an equivalence
class. The \( C_6 \) equivalence class contains 3 data cells, but
the quotient cube needs to retain its upper and lower bounds when
storing the equivalence class, so its occupied storage space is
3 data cells, and its query capacity is also 3. However the
\( C_2 \) equivalence class has only two data cells as upper and
lower bounds, while its query capacity contains 11 data cells.

Compared to the \( C_2 \) equivalence class, \( C_6 \) is considered to be
a relatively “small” equivalence class with relatively small
compression efficiency and query answering ability, while
\( C_2 \) has relatively large compression efficiency and query
answering ability.

To solve the aforementioned problem, the reduced quotient
model is proposed in this paper. Inside the quotient cube,
there are equivalence classes of different sizes, and the size
of the equivalence classes is critical to their effective storage.
Unlike other data cube models, this paper takes the quotient
cube equivalence class as the starting point to further improve
the storage efficiency of the quotient cube equivalence class,
and we first introduce the equivalence class cover capacity \( Ca \),
which refers to the number of data cells covered or
contained by the equivalence class. And the data cube is
compressed by filtering and storing the equivalence classes
with a larger \( Ca \) value. In addition, a query algorithm based
on the reduced quotient cube model is proposed to enhance
the query capacity by using the coverage relation in the
equivalence classes.

This paper makes the following main contributions.

- We present the \( Ca \) and introduce the relevant definitions
  and theorems of the reduced quotient cube, as well as the
  definition of its data structure and construction method.
- Based on the observation of the structure of the reduced
  quotient cube, we conclude the calculation formula of the
  \( Ca \) value of equivalence classes, and through the
  statistical experiment on the distribution of the total
  value of \( Ca \) in the equivalence class of the quotient
cube, the fact that a large number of equivalence classes
  with small cover capacity does exist inside the quo-
tient cube is verified, and the result that the number of
equivalence classes with small \( Ca \) values will increase
sharply with the increase of dimension of data sets is
also discovered. Comparing the sum of the equivalence
class \( Ca \) before and after effective compression calcula-
tion between the quotient cube model and the reduced
quotient cube model demonstrates that the reduced quotient cube model after effective compression calculation is more efficient in terms of storage.

- Finally, the query algorithm of the reduced quotient cube model is designed based on the cover relation.

Through experiments, the query correctness of the reduced quotient cube model was verified, and the higher query efficiency of the reduced quotient cube model was demonstrated by comparing it to the traditional quotient cube model.

The paper is structured as follows: The background and research motivation are given in Section 1. Section 2 reviews the main concepts and essential properties of the data cube, particularly the quotient cube. A reduced quotient cube method based on equivalence class cover capacity is introduced in Section 3. In Section 4, the calculation formula of the Ca value of equivalence classes is discussed. Then we propose the effective compression calculation method of the reduced quotient cube and the query algorithm of the reduced quotient cube model. Section 5 describes the analysis of experimental simulations based on existing public data sets. Finally, our work is summarized in the last sections.

II. PRELIMINARY

The data cube proposed by Gray et al. [21] is the core data model for data warehouse OLAP. It allows data to be modeled and analyzed intuitively in the semantics of multiple analytic dimensions. OLAP provides a set of operations on the data cube, such as roll-up and drill-down operation, slice and dice operation, and rotate operation [22]. Although it is called “data cube”, it can be two-dimensional, three-dimensional, or higher [23], with each dimension representing an attribute in the data table.

The core operation of the data cube is to aggregate the attributes of multiple dimension sets on the data [24]. Each combination of dimensions is called cuboid/view, and a cuboid containing i attributes is called i-dimensional cuboid. If the data cube can be materialized (pre-computed) completely, it will be beneficial to subsequent queries on the data cube. However, the direct calculation and storage of the data cube without processing will result in much overhead. If the data cube has n dimensions, the total number of cuboids it contains is: \( N = \prod_{i=1}^{n} (L_i + 1) \), where \( L_i \) is the number of level of the i-th dimension.

In 2002, Lakshmanan et al. [13] first proposed the quotient cube: a data cube compression model, and pointed out that the quotient cube constructed based on cover relation \( \equiv_{cov} \) has unique properties, that is, all data cells in each cover equivalence class have the same aggregate value for any aggregate function. The quotient cubes discussed in this paper are all quotient cover cubes [25]. Some definitions are given below.

Definition 1 (Cover Relation): If the data cells \( u(a_1, \ldots, a_q) \in C \) and \( v(b_1, b_2, \ldots, b_q) \in C \) satisfy the following two conditions at the same time, then it is called \( u \) covering \( v \) (\( u \supseteq v \)) or \( v \) is covered by \( u \) (\( v \subseteq u \)), where \( C \) is the data cube. The cover relation is a binary relation:

1) For all \( a_i, b_i \in C \), if \( u(a_i) \neq \ast \), then \( u(a_i) = v(b_i) \).
2) For all \( a_i, b_i \in C \), if \( a_i = \ast \), then \( v(b_i) \) is any.

If \( v \subseteq u \), we can say that \( u \) generalizes \( v \), or \( v \) specializes \( u \). In other words, \( u \) drills down to \( v \) or \( v \) rolls up to \( u \).

Definition 2 (Base Tuple Set): Given a base table \( R \), and the data cube \( C \) of \( R \), the base tuple set of a data cell \( c \in C \) is \( BTS(c) = \{ t | t \in R \text{ and } t \leq c \} \), that is the set of all base tuples that can be rolled up to the data cell \( c \), or the set of all base tuples that can be covered by \( c \).

Definition 3 (Cover Equivalence): If the data cells \( u \in C \) and \( v \in C \) satisfy \( BTS(u) = BTS(v) \), then \( u \) and \( v \) are said to satisfy the cover equivalence, denoted by \( u \equiv v \).

Definition 4 (Equivalence Class): An equivalence class is a set of data cells that have the same base tuple set. For a given data cell \( u \) and \( v \), if \( u \equiv v \), then \( u \) and \( v \) belong to the same equivalence class. Lemma 1 will prove that the measure values of data cells belonging to the same equivalence class must be equal.

Definition 5 (Upper and Lower Bounds): The number of non-asterisk dimensions of the data cell is recorded as the layer number of the data cell. For data cells in the same equivalence class, the maximum layer number is called the upper bound, and the minimum layer number is called the lower bound. When the equivalence class contains only one data cell, the upper and lower bounds may not be distinguished, or the upper and lower bounds are the same.

Definition 6 (Partial Order Relation): Given data cells \( u \in C, v \in C \), where \( C \) is the data cube. If \( u \) and \( v \) have a cover relation, it is also reflexive, antisymmetric and transitive between \( u \) and \( v \), then \( u, v \) have a partial order relation.

Definition 7 (Lattice): Let \( (L, \preceq) \) be a lattice, then \( (L, \preceq, \preceq_{cov}, \preceq_c) \) is a quotient cube, where \( \preceq_{cov} \) is the equivalence class defined as follows:

\[ \equiv_{cov} = \{ < c_1, c_2 > | c_1, c_2 \in (L, \preceq), BTS(c_1) = BTS(c_2) \} \] (1)

Lemma 1: If data cells \( u \in C \) and \( v \in C \) satisfy \( u \equiv v \), then the measure values of \( u \) and \( v \) are equal.

Proof: We can infer \( BTS(u) = BTS(v) \) in terms of \( u \equiv v \).

Since the base tuple sets of \( u \) and \( v \) are the same, the measure values calculated by the same aggregation function must be the same.

Lemma 2: The equivalence class is a convex set.

Proof: Given data cells \( u, v, w \in C \), assuming that \( u \leq w \leq v \), \( u \equiv v \). We can infer that \( BTS(u) \subseteq BTS(w) \subseteq BTS(v) \) in terms of \( u \leq w \leq v \), since \( u \equiv v \), then \( BTS(u) = BTS(v) \), thus \( BTS(w) = BTS(u) = BTS(v) \), \( w \equiv u \equiv v \).

Lemma 3: Suppose \( t_1 \) be a data cell and \( t_2 \) be an upper bound cell. If \( t_1 \) covers \( t_2 \) and among all the covered
upper bound cells, the layer of \( t_2 \) is closest to \( t_1 \), then 
\[ \text{BTS}(t_1) = \text{BTS}(t_2). \]

### III. REDUCED QUOTIENT CUBE MODEL

In the quotient cube lattice, there is a cover relation between different data cells, and the set of data cells formed by the cover relation is called the equivalence class. The measure values of data cells of the same equivalence class must be equal, so only the upper bound of the equivalence class needs to be saved, and the query of other data cells can be obtained from the cover relation between them, so the equivalence class can not only retain the semantics but also compress the data.

#### FIGURE 2. Complete upper bound lattice structure.

According to the above characteristics of the quotient cube equivalence class, this paper takes the quotient cube equivalence class as the starting point to explore methods to improve further the storage efficiency of the quotient cube equivalence class. In this regard, we first need to make a relevant analysis of the lattice structure of the quotient cube equivalence class. By observing the lattice structure in Fig. 2, it is found that the equivalence class is a complete partition of the whole lattice, and the upper bound of the equivalence class is also a complete partition of the partial lattice. Since the equivalence class is a convex set, the study of the equivalence class can be transformed into the study of the upper bound of the equivalence class, which is also helpful to further compress the data cube by analyzing its lattice structure.

In this paper, we propose a reduced quotient cube model, in which only the upper bound of the equivalence class in the data cube is preserved, and each upper bound is covered by any other upper bound or covers other upper bound in the model. Hence, the reduced compression model has a complete upper bound lattice structure. Some lattice structures are defined as follows:

**Definition 9 (Path):** If there is a cover relation between data cell \( u \) and data cell \( v \), it is believed that there is a path between \( u \) and \( v \), expressed as \( \text{Path}(u \rightarrow v) \).

**Definition 10 (Dangling Cell):** If there is no path from the current cell to the base cell, it is called a dangling cell.

**Definition 11 (ALL Path):** The path from any cell \( u \) to the ALL-cell is called the ALL path, expressed as \( \text{Path}(u \rightarrow \ast) \).

**Definition 12 (Cover Capacity):** The number of all data cells covered within an equivalence class is called the cover capacity of the equivalence class. By analyzing the upper bound lattice structure of equivalence classes, the \( Ca \) values of each upper bound can be calculated by the following methods:

\[ Ca = 2^{(l_u)} - \sum_{i=1}^{u-1} Ca_i \]  

where \( l_u \) is the layer number of the current data cell \( u \), \( Ca_i \) is the \( Ca \) value of the \( i \)-th data cell on the path from the ALL-cell to the data cell \( u \).

#### FIGURE 3. Incomplete upper bound lattice structure.

As depicted in Fig. 2, from the data cell \( \ast, \ast, \ast, \ast \), there is only one data cell in the path from the All-cell to itself, so the \( Ca \) value of the data cell \( \ast, \ast, \ast, \ast \) is: \( Ca = 2^0 = 1 \)

Starting from the All-cell, there are two data cells (All-cell and itself) on the path to the data cell \( \ast, \ast, \ast, P_2 \), so the \( Ca \) value of the data cell \( \ast, \ast, \ast, P_2 \) is: \( Ca = 2^1 - 2^0 = 2 - 1 = 1 \).

The \( Ca \) value of the base cell \( Q_1, R_2, S_1, P_2 \) can also be calculated. Starting from the All-cell, there are four data cells \( (\ast, \ast, \ast, \ast), (\ast, \ast, \ast, P_2), (Q_1, \ast, S_1, \ast) \) and \( (Q_1, R_2, S_1, P_2) \) on the path to the base cell \( Q_1, R_2, S_1, P_2 \), so the \( Ca \) value of the base cell \( Q_1, R_2, S_1, P_2 \) is: \( Ca = 2^4 - (Ca_1 + Ca_2 + Ca_3) = 16 - (1 + 1 + 3) = 11 \).

By analogy, the \( Ca \) value of the upper bound of all equivalence classes can be calculated.

**Definition 13 (Reduced Quotient Cube):** In quotient cube, there are \( n \) equivalence classes \( C = \{ c_1, c_2, \ldots, c_n \} \), whose occupied storage is \( S = \{ s_1, s_2, \ldots, s_n \} \), the corresponding cover capacity is \( Ca = \{ ca_1, ca_2, \ldots, ca_n \} \), given the limited space \( S \), the integer \( k \), and \( K \) equivalence classes are taken from \( n \) equivalence classes is called the reduced quotient cube if satisfies

\[ \sum_{i=1}^{k} s_i \leq S \]  

\[ L = \arg\max \left( \sum_{i=1}^{k} Ca_i \right) \]
It is an approximation of the quotient cube, which can further compress the space of quotient cube while maintaining a good query answering ability.

**Lemma 4**: If the data cell \( u \) covers the data cell \( v \), there must be no other path from \( u \) to \( v \).

**Lemma 5**: For any cell \( c \), there must be a path from the base cell to the cell \( c \).

By observing Fig. 2, the bottom \((*,*,*,*)\) is called ALL cube (cell), \((Q_1, R_2, S_1, P_2), (Q_1, R_1, S_1, P_1), (Q_2, R_1, S_2, P_1), \) and \((Q_2, R_1, S_2, P_2)\) are the base table of data tuples called base cells. The path from \((*,*,*,*)\) to the base cell \((Q_1, R_2, S_1, P_2)\) is called ALL path, which is expressed as Path\(((Q_1, R_2, S_1, P_2) \rightarrow *)\). In the Fig. 3, the data cell \((*,*,*, P_2)\) is called a dangling cell because there is no path to the base cell. According to the definition of cover capacity, the greater the \( Ca \) of the equivalence class, the more data cells covered by the equivalence class, the stronger the ability to answer queries. Based on the research of cover capacity, this paper considers how to retain the equivalence class with a large \( Ca \) value as much as possible in order to answer more queries in the limited storage space. The formal definition of this problem is given below:

**Problem definition 1**: Given a limited space with size of \( S \), solve the reduced quotient cube. That is, there are \( n \) equivalence classes \( C = \{c_1, c_2, \ldots, c_n\} \), the corresponding cover capacity is \( Ca = \{ca_1, c_2a_2, \ldots, c_na_n\} \), and the storage size is \( S = \{s_1, s_2, \ldots, s_n\} \), then find \( K \) equivalence classes \( c_i \in C \) satisfying \( \sum_{i=1}^{k} s_i \leq S \), making \( L = \arg\max \left\{ \sum_{i=1}^{k} ca_i \right\} \).

The measures of data cells in the same equivalence class are equal because the equivalence class is a convex set. Only the lattice structure of the upper bound of the equivalence class is preserved in the reduced quotient cube model. As a result, the cover capacity in the reduced quotient cube model is essentially the cover capacity of the model’s upper bound cell.

![Diagram](Q2,R1,S2,*)

\((Q2,R1,**,*)\) \((Q2,**,S2,*)\) \((*R1,S2,*\) \((Q2,**,*,*)\) \((**,S2,*)\) \((Q2,*,*,*)\) \((Q2,R1,**,*)\) \((Q2,*,S2,*)\) \((*,R1,S2,*)\) \((Q2,**,*,*)\) \((**,S2,*)\) \((Q2,*,*,*)\) \((Q2,R1,**,*)\)

\(Ca:6\)

**FIGURE 4.** An example of the upper bound cover capacity of equivalence class.

As shown in Fig. 4, the upper bound \((Q2, R_1, S_2, *)\) on the left is covered by six data cells. So in the reduced quotient cube model, the cover capacity of the upper bound \((Q2, R_1, S_2, *)\) is 6, and the number of data cell queries that can also be answered 6.

**Definition 14 (Submodular Function)**: If \( \Omega \) is a limited set, a submodular function is a set function \( f : 2^{\Omega} \rightarrow \mathbb{R} \), where \( 2^{\Omega} \) denotes the power set of \( \Omega \), which satisfies one of the following equivalent conditions.

1) For all \( A, B \subseteq \Omega \) with \( A \subseteq B \) and every \( s \in \Omega \setminus B \), we have that
   \[
   f(A \cup \{s\}) - f(A) \geq f(B \cup \{s\}) - f(B)
   \]
   \[\text{(5)}\]
2) For all \( A, B \subseteq \Omega \), we have that
   \[
   f(A) + f(B) \geq f(A \cup B) + f(A \cap B)
   \]
   \[\text{(6)}\]
3) For all \( A \subseteq \Omega \) and \( s, s_1, s_2 \in \Omega \setminus A \) such that \( s_1 \neq s_2 \), we have that
   \[
   f(A \cup \{s_1\}) + f(A \cup \{s_2\}) \geq f(A \cup \{s_1, s_2\}) - f(A)
   \]
   \[\text{(7)}\]

**Lemma 6**: The sum of \( Ca \) values that can be represented by the limited space satisfies the property of submodular.

**Proof**: First we have to prove the monotonicity: since equivalence classes are independent of each other, and their \( Ca \) is also independent, we can say that \( Ca(A+B) = Ca(A) + Ca(B) \) in the limited space, and then \( Ca(A \cup \{s\} - Ca(A) = Ca(A) + Ca(\{s\}) - Ca(A) = Ca(\{s\}) > 0 \), so the function \( Ca \) is monotonically increasing. Then, we need to prove the submodularity: for all \( A, B \subseteq C \), \( Ca(A \cup B) + Ca(A \cap B) = Ca(A) + Ca(B) - Ca(A \cap B) + Ca(A \cap B) = Ca(A) + Ca(B) \). In this case, it satisfies the formula: \( Ca(A) + Ca(B) \geq Ca(A \cap B) + Ca(A \cup B) \).

To sum up, the function \( Ca \) satisfies the submodularity, and the property can be used to solve the problem mentioned above.

**IV. REDUCED QUOTIENT CUBE ALGORITHM**

**A. ALGORITHM FRAMEWORK**

First, the cover capacity of all upper bound cells in the reduced quotient cube model is calculated according to the lattice structure of the upper bound of the equivalence class. Then, according to the cover capacity of the upper bound of each equivalence class, based on the idea of dynamic programming, the maximum value of the sum of \( Ca \) of the equivalence class in the limited space is calculated. Fig. 5 is the entire algorithm framework.

![Diagram](Algorithm Framework)

**FIGURE 5.** Algorithm framework.

**B. CONSTRUCTION ALGORITHM**

The construction process of the reduced quotient cube model is divided into two steps: the first step is to use the traditional quotient cube construction algorithm to generate a preliminary quotient cube (Algorithm 1). The second step is to search and traverse the dangling cell inside the quotient cube and use the cover relation between data cells to add,
delete, and modify the path to generate a reduced quotient cube (Algorithm 2).

**Algorithm 1 Generate Preliminary Quotient Cube**

**Input:** ALL cuboid, Base Table $Q$

**Output:** preliminary quotient cube structure (incomplete)

1. **DFS($cell$, partition, $k$, parentCell)**
   - /* cell: the cell to be decomposed, partition: base table partition, $k$: record the dimensions of data cell decomposition, parentCell: record the parent cell */
   - 1. Calculate the upper bound $ub$ of the equivalence class of the cell
   - 2. Record the upper bound $ub$, equivalence class cover capacity $Ca$, the parent cell of equivalence class parentCell, and the measure value $mea$ of the upper bound $ub$
   - 3. for each $0 < j < k$ in cell do
     - if (cell $[j] = ALL$ & & $ub [j] = ALL$) then
       - return ;
     - end if
   - 4. for each $k < j < n$ in cell do
     - if (cell $[j] = ALL$) then
       - for each partition value $v$ on $j$ dimension do
         - $ub [j] = v$
         - decomposed cell upper bound $ub$ to obtain a partition part
         - if part $\neq \emptyset$ then
           - DFS (c, part, $j$, ub)
         - end if
       - end for
     - end if
   - 9. for each $k < j < n$ in cell do
     - if (cell $[j] = ALL$) then
       - for each partition value $v$ on $j$ dimension do
         - $ub [j] = v$
         - decomposed cell upper bound $ub$ to obtain a partition part
         - if part $\neq \emptyset$ then
           - DFS (c, part, $j$, ub)
         - end if
       - end for
     - end if
   - 19. return

The traditional algorithm of quotient cube construction (based on DFS(Depth-First Search)) requires obtaining the attribute value set of each dimension first, and then starting from $ALL$-cell, using DFS to calculate the measure value of the upper bound of the equivalence class, which the current data cell belongs, and then decomposing the current dimension value from the upper bound data cell based on the base tuple set of the equivalence class, and searching the upper bound of the next equivalence class of the same dimension until all data cells are searched.

Table 1 shows a base table $R$ with four dimensions and one measure containing four tuples. First, the first dimension of $ALL$-cell is decomposed into two data cells $(Q1, *, *, *)$ and $(Q2, *, *, *)$, and search the upper bounds $(Q1, *, S1, *)$ and $(Q2, R1, S2, *)$ of the current data cells, and then decompose the next dimension of the upper bound of the data cells. According to this process, all dimensions of the current data cell are decomposed.

The traditional quotient cube construction algorithm uses the “jump” technique, which skips many unnecessary search processes when generating all the correct upper bounds of equivalence classes. Although the time and space complexity required by the algorithm are reduced, this “jump” technology cannot preserve the complete cover relation between the upper bounds of equivalence classes. As shown in Fig. 3, the figure shows the quotient cube constructed by the DFS-based quotient cube construction algorithm, without retaining the complete lattice structure (for example, it does not show the relation of $(Q1, R2, S1, P2)$ is covered by $(*, *, *, P2)$), and cannot generate the complete upper and lower bound relations. For the lattice structure shown on the right, the lattice can be expressed as a complete partition, and the equivalence class is a convex set, so only the upper bound needs to be considered when analyzing the equivalence class. The complete equivalence partition is better for studying its lattice structure in order to further compressing the data cube.

**Algorithm 2 Reduced Quotient Cube**

**Input:** preliminary quotient cube (incomplete), UpperBoundList

**Output:** reduced quotient cube and its corresponding $Ca$

1. RQC_Construct(UpperBoundList, BaseCellList, DanglingCellList)
2. for each baseCell in BaseCellList do
   3. for each danglingCell in DanglingCellList do
      4. if danglingCell cover baseCell then
         5. add danglingcell to the All path of this basecell
      6. end if
   7. end for
8. end for
9. return

**TABLE 1. The base table $R$.**

| Dimension 1 | Dimension 2 | Dimension 3 | Dimension 4 | Measurement |
|-------------|-------------|-------------|-------------|-------------|
| Q1          | R1          | S1          | P1          | 10          |
| Q2          | R1          | S2          | P2          | 20          |
| Q2          | R1          | S2          | P2          | 20          |

**FIGURE 6. The upper bound lattice structure of reduced quotient cube equivalence class constructed by algorithm 2.**

As shown in Fig. 6(a), the preliminary upper bound lattice structure of the quotient cube equivalence class is constructed through Algorithm 1. There is a cover...
relation between dangling cell \((*, R1, *, P1)\) and base cells \((Q1, R1, S1, P1), (Q2, R1, S2, P1)\). However, in the left figure, the three data cells are not linked, and the cover relation between \((*, R1, *, P1)\) and \((*, R1, *, *)\) is not shown. Using Algorithm 2, search for all dangling cells, compare them with the data cells on the ALL path of the base cells, find out the cover relations, and add, delete, and modify them. When the dangling cell \((*, R1, *, P1)\) is searched, it is compared with all data cells on the path of the base cell \((Q1, R1, S1, P1)\), and it is found that \((*, R1, *, P1)\) covers \((Q1, R1, S1, P1)\), and is covered by \((*, R1, *, *)\), so the relation between them is added, and the relation between \((*, R1, *, P1)\) and \(Afi\)-cell is deleted according to Theorem 4. Thus, in Fig. 6(b), the reduced quotient cube structure can be obtained.

**C. COMPRESSION ALGORITHM**

The “jump” technology is used in the traditional quotient cube construction algorithm, and it is unable to construct the entire upper bound lattice structure of equivalence classes, making it impossible to correctly calculate the upper bound \(Ca\) value of each equivalence class. To solve this problem, the reduced quotient cube model is used to build the complete upper bound lattice structure of equivalence classes and correctly calculate the cover capacity of each upper bound.

According to Definition 12, the algorithm for \(Ca\) calculation is as follows (Algorithm 3):

**Algorithm 3 Calculate \(Ca\)**

**Input:** reduced quotient cube model  
**Output:** \(Ca\) of upper bound  
1: \text{RQC\_Ca(cell, temp, parents)} \ //cell: current data cell, \(temp\): sum of other data cell \(Ca\) on the path, parents: father node of cell  
2: \(Ca = 2^n - temp;\)  
3: \(temp = temp + Ca;\)  
4: record \(Ca\) of cell;  
5: \text{for each } 0 < j < n \text{ in parents do}  
6: \text{if (cell == parents (j)) then}  
7: \text{continue;}  
8: \text{call RQC\_Ca(parents(j), tempparents(j), parents)}  
9: \text{end if}  
10: \text{end for}  
11: \text{return}

The different distributions of the equivalence class \(Ca\) can be obtained by calculating the cover capacity of the upper bound of each equivalence class. In Fig. 2, the cover capacity of \((Q1, R2, S1, P2)\) is 11, and the cover capacity of \((*, *, *, P2)\) is 1, and the cover capacity of equivalence classes is different. Due to the limited storage space, when storing equivalence classes, it is expected that the greater the cover capacity of equivalence classes, the better.

In Fig. 7, the circle represents the data cube equivalence class stored in a limited space. The number inside the circle means the cover capacity of the equivalence class, and the size of the circle also represents the cover capacity. Because the equivalence class’s cover capacity is not considered before compression, putting an equivalence class with a small cover capacity into the storage space will result in a lot of waste. However, after compression, according to the required storage space size and the cover capacity of the equivalence class, the equivalence classes with large cover capacity are prioritized for storage. In this way, after effective calculation, the data cube can further compress the storage space and answer the queries of most data cells.

Dynamic programming [26] is a commonly used mathematical method to solve the optimization problem of a multi-stage decision-making process, and the effective compression calculation of the data cube is to use a dynamic programming algorithm. Some equivalence classes with high \(Ca\) value but small storage space are selected to make the sum of \(Ca\) value of equivalence classes as large as possible in the limited storage space. That is to say, the ability to answer queries of equivalence classes should be as high as possible. The effective compression calculation algorithm is as follows (Algorithm 4):

**Algorithm 4 Effective Compression Calculation \(Ca\)**

**Input:** \text{size} (the current storage space); \(n\) (the number of upper bounds of the equivalence class); \text{uSize} (the space required for the storage of the current equivalence class); \text{uCa} (the cover capacity of the equivalence class)  
**Output:** Maximum query capacity of equivalence class  
1: \text{for each } 0 < i < n \text{ in uSize[i]} do  
2: \(j = \text{size};\)  
3: \text{for each } 0 <= j do  
4: \text{reverse order implementation}  
5: \text{MaxCa}[j] = \text{MaxCa}[j-uSize[i-1]]+uCa[i-1], \text{MaxCa}[j])  
6: \text{end for}  
7: \text{end for}  
8: \text{return MaxCa[\text{size}]}  

We use dynamic programming to get the optimal value. If \text{MaxCa} [\text{size}] is used to store the intermediate state variable, \text{MaxCa}[j] represents the maximum value of the total cover capacity of the first \(j\) equivalence classes in the storage space with a storage capacity of \(j\). Only the value of
MaxCa\(j = \text{size}\) needs to be known, that is, the maximum Ca value of the equivalence class in the limited storage space.

Considering the state transition equation of dynamic programming algorithm MaxCa\(i\), assuming that the maximum sum of Ca of the first \(i-1\) equivalence classes loaded into the storage space with storage capacity \(j\) is MaxCa\(i - 1\), and the current storage space capacity \(j\) is unchanged, the storage of the \(i\)-th equivalence class is discussed as follows:

The Ca value of the \(i\)-th equivalence class must be less than or equal to the capacity \(j\) to satisfy the condition:

- If \(uCa[i] > j\), the \(j\)-th equivalence class cannot be loaded into storage space with capacity of \(j\). Then,

\[
\text{MaxCa}[i] = \text{MaxCa}[i - 1] \quad (8)
\]

- If \(uCa[i] \leq j\), the equivalence class can be loaded into storage space with capacity of \(j\), then if the equivalence class is loaded, there will be

\[
\text{MaxCa}[i] = \text{MaxCa}[j - uCa[i - 1]] + \text{value}[i - 1] \quad (9)
\]

After the \(i\)-th equivalence class is loaded into the storage space with the capacity of \(j\), it is necessary to judge whether the sum of the equivalence classes Ca in the current storage space is the largest. The state transfer equation is as follows:

\[
\text{MaxCa}[i] = (\text{MaxCa}[i - 1] > (\text{MaxCa}[j - uCa[i - 1]]) \\
\quad + \text{value}[i - 1]) \text{MaxCa}[i - 1] : (\text{MaxCa}[j - uCa[i - 1]] \\
\quad + \text{value}[i - 1]) \quad (10)
\]

Finally, the final result can be obtained by reverse order implementation.

The greedy algorithm is commonly used to solve the submodular problem because it can obtain an approximate optimal solution, and the effective compression calculation of data cube is also can use a greedy algorithm. Combined with the submodular properties mentioned above, the greedy algorithm for effective compression is shown in Algorithm 5.

Algorithm 6 Query Algorithm

**Input:** query item

**Output:** query result

1. Sort(Upper bound set \(L(c)\))
2. /*traverse query item set*/
3. while query item \(q \in \text{query set}\) do
4. while upper bound cell \(c \in \text{upper bound set}\) do
5. if \(q\) cover current upper bound cell \(c\) then
6. \(\text{return}\) measure value of \(c\);
7. end if
8. end while
9. end while
10. return
Q. Wang et al.: Reduced Quotient Cube: Maximize Query Answering Capacity

Fig. 8, Fig. 9 and Fig. 10 show the experimental results. With the change of dimension, the number of equivalence classes with \( Ca \) value of 1 is increasing, and in the three figures, the number of equivalence classes with \( Ca \) value of 1 is the most. In the limited storage space, we do not expect this kind of equivalence class to appear in large quantities because its cover capacity is one and the storage space it requires is also 1, which does not have the effect of compression. In addition, with the increase of the number of dimensions, the value range of equivalence class \( Ca \) becomes larger; there exist some equivalence classes with large \( Ca \) values. Therefore, in the limited storage space, the equivalence class with a high \( Ca \) value should be selected for storage.

In addition, we selected the real data set Food-Mart and the synthetic data set TPC-H to store and calculate the quotient cube effectively. To construct the quotient cube, we took 13795 pieces of data and 10 dimensions from the Food-Mart data set, as well as 100,000 pieces of data and 5 dimensions from TPC-H. The dimensions, measure values, and base table size of the data set are shown in Table 2 and Table 3. In the experiment, we also simulate the related operations of the database in the decision support system.

Using the above data sets, we constructed the Food Mart quotient cube TPC-H quotient cube. The total number of equivalence classes of the Food Mart quotient cube is 27933, the total \( Ca \) value of all equivalence classes is 1348661, and the storage space required for equivalence classes is 121921. The total number of equivalence classes of the TPC-H quotient cube is 193179, the total \( Ca \) value of all equivalence classes is 2535305, and the storage space required for all equivalence classes is 882189.

After effective compression calculation of the quotient cube and reduced quotient cube models, it can obtain the change of the sum of equivalence classes \( Ca \) in different storage spaces, and the storage spaces selected in the experiment ranged from 1000 to 80,000. Fig. 11 is a line chart which that depicts the change of the sum of the equivalence class \( Ca \) of the Food Mart data set based on the reduced quotient cube model after effective compression calculation. When the storage space is 15000, the total \( Ca \) of all equivalence classes is 1326295. Therefore, when the limited storage space is 15000, its query ability is about 98.3% of the maximum, but its storage space is only 12.3% of the maximum.

Fig. 12 is a line chart which that depicts the change of the sum of the equivalence class \( Ca \) of the TPC-H data set based on the reduced quotient cube model after effective compression calculation. When the storage space is 100,000, the equivalence class in the storage space can answer 2431723 data cell queries, so when the limited storage space size is 100,000, its query ability is about 95.9% of the maximum, but its storage space is the only 11.3% of the maximum.

**TABLE 2. Food mart dataset.**

| Dimensions Selection | Measurement Selection | Base Table Size |
|----------------------|-----------------------|-----------------|
| store_state, store_city, store_name, store_squ, store_type, month_of_year, media_type, promotion_name, education,yearly_income | cell_sales | 13795 |

**TABLE 3. TPC-H dataset.**

| Dimensions Selection | Measurement Selection | Base Table Size |
|----------------------|-----------------------|-----------------|
| l_orderkey,l_partkey, l_suppkey,l_linenumber, l_quantity | l_extendedprice | 100000 |

**FIGURE 11. Food-Mart dataset.**

**FIGURE 8. Dataset1 (5 dimensions) and dataset2 (6 dimensions).**

**FIGURE 9. Dataset3 (7 dimensions).**

**FIGURE 10. Dataset4 (8 dimensions).**

**FIGURE 12. Dataset5 (9 dimensions).**
Q. Wang et al.: Reduced Quotient Cube: Maximize Query Answering Capacity

Fig. 12, TPC-H dataset.

Fig. 13 and Fig. 14 show the comparison of the sum of equivalence classes $Ca$ before and after calculation by using the quotient cube model (QC) and after calculation by using the reduced quotient cube model (RQC), when the data sets are Food-Mart and TPC-H, respectively. In general, for the two data sets, under the same limited storage space, the sum of the equivalence classes $Ca$ of the reduced quotient cube model after effective storage calculation is the highest.

Fig. 13, Food-Mart dataset.

According to the experimental results, we can conclude that in the same storage space, the query answer ability of the reduced quotient cube to data tuples is always larger than the quotient cube. Therefore, to better illustrate the improvement of the storage performance of the data cube by the reduced quotient cube model, the number of data cells covered by the equivalence class is taken as the description index of the compression ratio of the reduced quotient cube model. Fig. 15 and Fig. 16 show the change of the compression ratio of the reduced quotient cube in different limited spaces in two data sets. With the increase of the limited storage space, the compression ratio of the reduced quotient cube is decreasing. The reduced quotient cube will first store the equivalence classes with large $Ca$ value, and the remaining equivalence classes with small $Ca$ value will be more and more, and the growth trend of the sum of equivalence classes $Ca$ will also slow down, and the compression efficiency will also decrease. Therefore, the smaller the storage space, the more pronounced the compression efficiency of the reduced quotient cube model.

Fig. 14, TPC-H dataset.

B. QUERY PERFORMANCE OF REDUCED QUOTIENT CUBE MODEL

We used the Food-Mart data set to test the query performance of the quotient cube and the reduced quotient cube models, and we ran 5000, 10000, 15000, 20000, and 25000 queries on it, respectively. As shown in Fig. 17, when the number
of queries is the same, the query performance of the reduced quotient cube model is nearly twice that of the quotient cube. This is because the query item in the quotient cube needs to cover judgment of the upper bound of the equivalence class and the lower bound. In the reduced quotient cube model, the query item only needs to consider whether it covers the current upper bound cell to respond to the query. Therefore, the query performance of the reduced quotient cube model is better than that of the traditional quotient cube model.

VI. CONCLUSION AND FUTURE WORK

A. CONCLUSION

The cover capacity of the equivalence class in quotient cubes varies, and there will be a problem of less efficient storage in the limited storage space. In order to find the maximum value of $C_a$ in the equivalence class in the limited space, we further investigate the lattice structure of the quotient cube and find a way to allow the equivalence classes of the quotient cube to be stored efficiently. This paper proposes and implements a reduced quotient cube model, which retains the complete equivalence class upper bound lattice structure of the quotient cube. We verified the fact that there are a large number of equivalence classes with small cover capacity in the quotient cube, and the number of equivalence classes with a small $C_a$ value will increase sharply with the increase of dimension of the data set. Furthermore, it has been demonstrated through comparison experiments that the reduced quotient cube model after the effective calculation is efficient in terms of storage efficiency and query performance.

B. FUTURE WORK

Although we present the effectively reduced quotient cube and its query ability (cover capacity), the work of this paper is still worthy of thorough study. The data cube is stored in the form of equivalence class after efficient computation, and it is hoped that a better storage structure for storing data cubes can be found in the future work to achieve further improvements in storage efficiency. Besides, in a limited space, although the quotient cube can make the total cover capacity of the equivalence class reach the maximum after effective calculation, it does not consider the user’s query habits. Therefore, in the next work, we intend to combine RQC with user query patterns and corresponding replacement algorithms such as the LRU (Least Recently Used) or LFU (Least Frequently Used), which consider the temporal or spatial locality of query data.

REFERENCES

[1] D. J. Dewitt, S. Madden, and M. Stonebraker, “How to build a high-performance data warehouse.” Tech. Rep., 2006.
[2] U. Alifah and G. F. Siddique, “Big data augmentation with data warehouse: A survey,” in Proc. IEEE Int. Conf. Big Data (Big Data), Dec. 2018, pp. 2785–2794.
[3] S. O. Salinas and A. C. Lemus, “Data warehouse and big data integration,” Int. J. Comput. Sci. Inf. Techn., vol. 9, no. 2, pp. 1–17, 2017.
[4] B. Reiner, K. Hahn, G. Hofling, and P. Baumann, “Hierarchical storage support and management for large-scale multidimensional array database management systems,” Tech. Rep., 2002, pp. 689–700.
[5] S. Chaudhuri and U. Dayal, “An overview of data warehousing and OLAP technology,” ACM SIGMOD Rec., vol. 26, no. 1, pp. 65–74, 1997.
[6] S. Cohen, “Containment of aggregate queries,” ACM SIGMOD Rec., vol. 34, no. 1, pp. 77–85, Mar. 2005.
[7] E. Galli, M. Gollarelli, S. Rizzi, A. Abelló, and O. Romero, Interactive Multidimensional Modeling of Linked Data for Exploratory OLAP, vol. 77. Amsterdam, The Netherlands: Elsevier, 2018, pp. 86–104.
[8] J. Gray, A. Bosworth, A. Lyaman, and H. Pirahesh, “Data cube: A relational aggregation operator generalizing GROUP-BY, CROSS-TAB, and SUB-TOTALS,” in Proc. 12th Int. Conf. Data Eng., Feb. 1996, pp. 1–17.
[9] Y. Li, C. Yu, and Z. Peng, “Relationship of data warehouse, OLAP and data mining.” Sci. Technol. Rastrous Steel Group Corp., vol. 31, no. 5, Oct. 2005.
[10] A. Gupta and I. S. Mumick, Materialized Views: Techniques, Implementations, and Applications. Cambridge, MA, USA: MIT Press, 1999.
[11] L. Q. Han, “Hadoop OLAP engine,” Tech. Rep., 2015.
[12] R. Tardio, A. Mate, and J. Trujillo, “A new big data benchmark for OLAP cube design using data pre-aggregation techniques,” Appl. Sci., vol. 10, no. 23, p. 8674, Dec. 2020.
[13] L. V. S. Lakshmanan, P. Jian, and J. Han, “Quotient cube: How to summarize the semantics of a data cube.” Tech. Rep., 2002.
[14] L. V. S. Lakshmanan, J. Pei, and Y. Zhao, “QC-trees: An efficient summary structure for semantic OLAP.” in Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD), 2003, pp. 64–75.
[15] L. Sheng-en and W. Shan. “Research on closed data cube technology,” Tech. Rep., 2004.
[16] W. Wang, J. Feng, H. Lu, and J. X. Yu, “Condensed cube: An effective approach to reducing data cube size,” in Proc. 18th Int. Conf. Data Eng., Apr. 2002, pp. 155–165.
[17] Y. Sismanis, A. Deligiannakis, N. Roussopoulos, and Y. Kotidis, “Dwarf: Shrinking the PetaCube,” in Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD), 2002, pp. 464–475.
[18] E. Garneau, S. Maabou, and M. Mosbah, “Functional dependencies are helpful for partial materialization of data cubes,” Ann. Math. Artif. Intell., vol. 73, nos. 1–2, pp. 245–274, Jan. 2015.
[19] Q. Zhao, Y. Zhu, D. Wan, and S. Tang, “A closed Frag-Shells cubing algorithm on high dimensional and non-hierarchical data sets,” in Proc. 12th Int. Conf. Ubiquitous Inf. Manage. Commun., Jan. 2018, pp. 1–8.
[20] K. Li and G. Li, “Approximate query processing: What is new and where to go?” Data Sci. Eng., vol. 3, no. 4, pp. 379–397, 2018.
[21] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh, “Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals,” Data Mining Knowl. Discovery, vol. 1, no. 1, pp. 29–53, Mar. 1997.
[22] B. Yang, J. Li, and H. Gao, “Query processing parallel data warehouse management system PDWMS,” in Proc. Nat. Database Conf. China, 2000.
[23] K. Beyer and R. Ramakrishnan, “Bottom-up computation of sparse and iceberg CUBE,” in Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD), 1999, pp. 359–370.
[24] J. You, J. Xi, P. Zhang, and H. Chen, “A parallel algorithm for closed cube computation,” in Proc. 7th IEEE/ACIS Int. Conf. Comput. Inf. Sci. (ICIS), May 2008, pp. 95–99.
[25] Y. Kotidis and N. Roussopoulos, “Dynamat: A dynamic view management system for data warehouses,” ACM Sigmod Rec., vol. 28, no. 2, pp. 371–382, 1999.
[26] R. E. Bellman and S. E. Dreyfus, Dynamic Programming. New York, NY, USA: Dover, 2003.

QUANKUN WANG received the B.E. degree in software engineering from Nanyang Institute of Technology, Nanyang, China. He is currently pursuing the M.E. degree with Kunming University of Science and Technology, Kunming, China. His current research interests include database and cloud computing.
JINGUO YOU received the M.Sc. degree from Kunming University of Science and Technology, Kunming, China, in 2005, and the Ph.D. degree from South China University of Technology, Guangzhou, China, in 2008. He is currently an Associate Professor with the Faculty of Information Engineering and Automation, Kunming University of Science and Technology. His interests include big data, data warehousing, and data mining. He is a Senior Member of China Computer Federation (CCF), a member of the Technical Committee of Information System, CCF, and a Communication Member of the Technical Committee of Database, CCF.

BENYUAN ZOU received the B.E. degree in network engineering from Shandong University of Science and Technology, Tai’an, China. He is currently pursuing the M.E. degree with Kunming University of Science and Technology, Kunming, China. His current research interests include database and machine learning.

YU CHEN received the B.E. degree in network engineering from Chongqing University of Posts and Telecommunications, Chongqing, China. He is currently pursuing the M.E. degree with Kunming University of Science and Technology, Kunming, China. His current research interests include database and machine learning.

XINGRUI HUANG received the B.E. degree in software engineering from Henan University of Technology, Zhengzhou, China. He is currently pursuing the M.E. degree with Kunming University of Science and Technology, Kunming, China. His current research interests include database and machine learning.

LIANYIN JIA received the Ph.D. degree from South China University of Technology, Guangzhou, China, in 2013. He is currently an Associate Professor with the Faculty of Information Engineering and Automation, Kunming University of Science and Technology, Kunming, China. His current research interests include databases, data mining, information retrieval, and parallel computing.

* * *