Name of workshop:
Workshop on Database Systems and Software Engineering

Title of the paper:

Flexible Data Cube for Range-Sum Queries in Dynamic OLAP Data Cubes

Abstract:

The data cube is frequently adopted to implement On-Line Analytical Processing (OLAP) and provides aggregate information to support the analysis of contents of databases and data warehouses. Range-sum queries require accessing large data cubes and adding the contents of massive cells immediately. Techniques have thus been proposed to accelerate range-sum queries by applying pre-aggregated specified cells of data cubes. However, this faster response leads to higher update cost and possible need for extra memory space overhead to store these pre-aggregated values. To offer the best trade-off choice, this study presents a novel approach that considers both query cost and update cost to construct data cubes. In addition, we shall also construct a space-optimal high-dimensional data cube and propose a flexible and optimal framework, called the Flexible Data Cube (FDC) approach. The FDC method can support a model to select or integrate these pre-aggregation techniques for each dimension.

Name, current affiliation, postal address, e-mail address, telephone number, and fax number for each author:

Chien-I Lee,

Addr: Institute of Information Education, National Tainan Teachers College Tainan, Taiwan 700, R.O.C.

E-mail: leeci@ipx.ntntc.edu.tw

Tel: (886)6-2133111 ext. 777

Fax: (886)6-2144409

Yu-Chiang Li,

Addr: Department of Computer Science and Information Engineering, National Chung Cheng University, Chiayi, Taiwan 621, R.O.C.

E-mail: lyc002@seed.net.tw

Tel: (886)5-2720411 ext. 23134

Name of the contact author: Chien-I Lee

Tel: (886)6-2133111 ext. 772

Fax: (886)6-2144409

Keywords: OLAP, Range-sum query, Data cube, Multidimensional database, Prefix sum.

Flexible Data Cube for Range-Sum Queries in Dynamic OLAP Data Cubes

Chien-I Lee[1], Yu-Chiang Li[2]

[1]Institute of Computer Science and Information Education, National Taiwan Teachers College, Tainan, Taiwan 700, R.O.C.
leeci@ipx.ntntc.edu.tw
[2]Department of Computer Science and Information Engineering, National Chung Cheng University, Chiayi, Taiwan 621, R.O.C.
lyc002@seed.net.tw

Abstract. The data cube is frequently adopted to implement On-Line Analytical Processing (OLAP) and provides aggregate information to support the analysis of contents of databases and data warehouses. Range-sum queries require accessing large data cubes and adding the contents of massive cells immediately. Techniques have thus been proposed to accelerate range-sum queries by applying pre-aggregated specified cells of data cubes. However, this faster response leads to higher update cost and possible need for extra memory space overhead to store these pre-aggregated values. To offer the best trade-off choice, this study presents a novel approach that considers both query cost and update cost to construct data cubes. In addition, we shall also construct a space-optimal high-dimensional data cube and propose a flexible and optimal framework, called the Flexible Data Cube (FDC) approach. The FDC method can support a model to select or integrate these pre-aggregation techniques for each dimension.

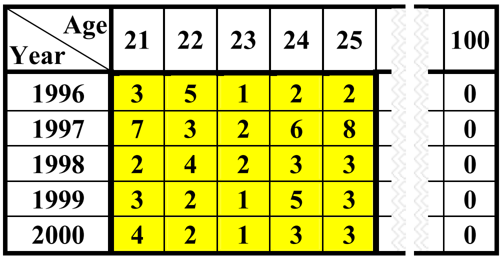
Keywords: OLAP, Range-sum query, Data cube, Multidimensional database, Prefix sum.

1 Introduction

Recent developments in information science have resulted in surprisingly rapid production of data. Efficiently managing massive data, quickly obtaining information and making correct decisions are very important. For example, the invention of data mining techniques has made the once impossible tasks of discovering and gathering hidden but potentially useful information from very large sets of data a part of our lives. The use of data mining procedures to obtain helpful information in data warehouses is currently an attention-drawing field of research. For business entities and government organizations to stay competitive in this era of information, it is of especially great importance to be able to quickly obtain integrated information from a data warehouse via On-Line Analytical Processing (OLAP) [4] to help the decision makers make quick and accurate decisions.

A data cube [9], or multi-dimensional database (MDDB) [1, 14], is a popular structure used in OLAP to support the interactive analysis of database and data warehouse. Typically, the data cube is implemented as a multidimensional array. A data cube is constructed from a subset of attributes in the database. The values of some particular attributes are chosen as *measure attributes*, while others are the *dimensions* (or *functional attributes*) of the data cube. The values of measure attributes are called *cells* and are aggregated according to the dimensions.

For example, consider a data cube maintained by a car-sales company. Assume that the figure of sales is of interest: the measure attribute is SALE_VOLUME, while YEAR and AGE_OF_CUSTOMER are dimensions. Let the domain of YEAR be 1996 to 2000, and let the domain of AGE_OF_CUSTOMER be 21 to 100. The data cube thus has 400(=5×80) cells, each of which contains a SALES_VOLUME value for the corresponding combination of two dimensions. Fig. 1 depicts the data cube. Consider a typical query, such as "*find the total sales quantity for all five years for customers of age between 21 and 25*," that is very important to decision-makers. A data cube should efficiently aggregate the values of cells within the range of the query, called the *range-sum query*. Clearly, 25(=5×5) cells must be accessed within the range, and the cells must be added up immediately. Intuitively, a larger query region yields a slower response.



Age Year	21	22	23	24	25	100
1996	3	5	1	2	2	0
1997	7	3	2	6	8	0
1998	2	4	2	3	3	0
1999	3	2	1	5	3	0
2000	4	2	1	3	3	0

Fig. 1. The original data cube of car sales

Analysts always expect timely answers to their queries. Accordingly, Geffner *et al.* proposed the *Group-By* approach [9] to shorten the response time. The method pre-store aggregated values of the measure attributes in all the cells along the dimensions. For the same example as shown in Fig. 1, Fig. 2 presents the extension of the data cube to 486(=6×81) cells by the Group-By approach. The approach needs only to access five cells in the shadowed area of Fig. 2. Nevertheless, the data cube with those pre-aggregated cells cannot efficiently support any range-sum query. For example, the other range-sum query, "*find the total quantity of sales for the first three years for customers aged between 21 and 25*," requires accessing 15(=3×5) cells. Furthermore, this data cube with pre-aggregated cells requires extra storage space and access to more cells when the values of the cells get updated. For example, as shown in Fig. 2, when the cell marked "*" is updated, the cell marked "+" also needs to be updated, and the reason is that the value of the cell marked "+" is the aggregate value of the five cells above it, including the updated cell marked "*". In this sense, applying pre-aggregation is always a trade-off between a faster response and a lower update overhead. Consequently, to provide a useful model to reach an appropriate balance is an important research topic in the field of OLAP.

Several approaches have been proposed, such as the PS method [11], the RPS method [6], the HC method [2] and the DDC method [5, 7], among others. As stated above, the pre-computed and stored aggregate values in some specified cells in the data cube are used to accelerate the answers to queries; however, at the same time, they slow down the update speed and require further space overhead. In this paper, we shall propose a new approach that takes both the query cost and the update cost into account to construct an optimal data cube. Moreover, we shall also briefly introduce the *Iterative Data Cubes* (*IDC*) approach [16, 18], which provides a modular framework for integrating the previously mentioned methods by selecting a pre-aggregation method for each dimension to construct a spatially optimal high-dimensional data cube, though the IDC method fails to consider the construction of a suitable data cube in certain query (or update) ratio.

Age **Year**	**21**	**22**	**23**	**24**	**25**	**100**	**Total**
1996	***3**	**5**	**1**	**2**	**2**	**0**	**254**
1997	**7**	**3**	**2**	**6**	**8**	**0**	**261**
1998	**2**	**4**	**2**	**3**	**3**	**0**	**305**
1999	**3**	**2**	**1**	**5**	**3**	**0**	**299**
2000	**4**	**2**	**1**	**3**	**3**	**0**	**310**
Total	**$^{+}$19**	**16**	**7**	**19**	**19**	**0**	**1429**

Fig. 2. The aggregative data cube of car sales

This research also proposes a flexible framework, called the *Flexible Data Cubes* (*FDC*) method, to support a model to select or integrate these pre-aggregation techniques for each dimension. Our work focuses mainly on methods that do not require any extra space overhead for dense data cubes.

The rest of this paper is organized as follows. In Section 2, we shall introduce some related works on pre-aggregation data cubes for range-sum queries. Then, in Section 3, we shall analyze the query cost and the update cost so as to provide a model for constructing a suitable data cube in certain query (or update) ratio. In Section 4, we shall propose a flexible framework for integrating the existing pre-aggregation methods in a space-optimal way. Section 5 will address the performance of our new method under conditions of various dimensionalities or different query (or update) ratios. Finally, the conclusion and direction for future research will be in Section 6.

2 Related Works

Let $D = \{1, 2, ..., d\}$ denote the set of dimensions. A d-dimensional data cube with one measure attribute can be represented by a d-dimensional array A of size $n_1 \times n_2 \times n_3 \times ... \times n_d$, where $n_i \geq 2$ ($i \in D$). Each entry in array A is called a *cell*, and each cell includes the aggregate value of the measure attribute corresponding to a given point in the d-dimensional space. For simplicity without loss of generality, array A is assumed to have a starting index zero in each dimension, and each dimension has the same size n. The total size of array A is n^d. For the range-sum query, a naive method is to access and add up all the cells within in the range of the query. In the worst case, n^d cells are accessed to answer the range-sum query. Otherwise, only the cells to be updated are accessed.

Many techniques have been proposed for approximate [8, 10, 15, 21] and progressive [12, 17, 20, 23] evaluation of range-sum queries. For dense data cubes and exact queries, Ho *et al.* [11] first proposed the *prefix sum* (*PS*) method to improve the effectiveness of the range-sum query. The approach uses an additional data cube, called the *prefix sum cube*, which is denoted by array P, with the same number of dimensions and size as array A. Each cell in P is indexed by $(x_1, x_2, ..., x_d)$, and stores the sum of the super cube from cell (0, 0, ..., 0) to cell $(x_1, x_2, ..., x_d)$ in array A. Restated, the sum of the entire array A can be obtained in the last cell of P (i.e., $P[x_1, x_2, ..., x_d]$). Namely, for all $0 \leq x_i < n_i$ and $i \in D$, $P[x_1, x_2, ..., x_d] = \sum_{i_1=0}^{x_1}\sum_{i_2=0}^{x_2}\cdots\sum_{i_d=0}^{x_d} A[x_1, x_2, ..., x_d]$.

Fig. 3 illustrates the main idea of the PS method in two dimensions. The sum that corresponds to the region of a range-sum query can be determined by adding and subtracting the sums of other various regions until the region of interest is obtain. Thus, the PS method reduces the range-sum query to the problem of reading 2^d single individual cells in array *P* [11]. In Fig.3, “*” denotes the corresponding cells in array *P* that must be accessed. The PS method provides range-sum queries in constant time, regardless of the size of the data cube. In the worst case, updating the value of cell [0, 0, ..., 0] in the original array *A* requires that all the n^d cells in array *P* be updated. Array *A* can even be deleted to maintain the same total required storage space once *P* is computed.

Geffner *et al.* [6] developed an approach called the *relative prefix sum* (*RPS*) method to reduce the update time with only a constant query cost. The RPS method partitions the data cube into small chunks called the *relative prefix array* and uses additional *overlay boxes*. However, the RPS method produces a space overhead.

Liang, Wang and Orlowska [13] proposed the *double relative prefix sum* (*Double RPS*) method to further reduce the update time of the RPS method. However, the Double RPS method depends on more storage space than the RPS method and results in running time O($n^{1/3}$) for each range-sum query.

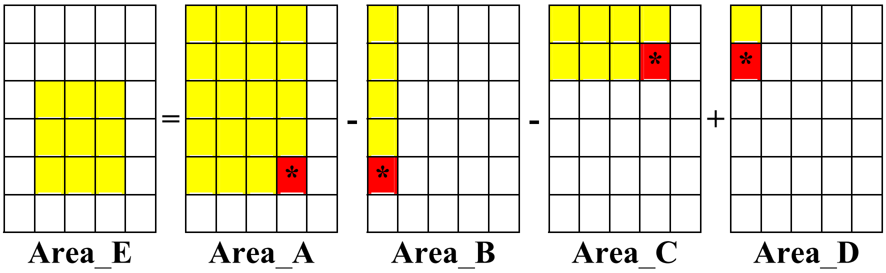


Fig. 3. A geometrical explanation for the two-dimension case: Sum(Area_E) = Sum(Area_A) - Sum(Area_B) - Sum(Area_C) + Sum(Area_D)

Chan and Ioannidis [2] specified the *hierarchical cubes* (*HC*) method based on two orthogonal dimensions. A particular cube, called the *hierarchical band cube* (*HBC*), exhibits a better trade-off between query and update than the RPS method. The index mapping structure, however, is too complicated.

Geffner *et al.* [5, 7] presented the *Dynamic Data Cube* (*DDC*) method. The DDC method balances the query cost and the update cost such that both time complexities are $O(log^d n)$. However, the DDC method results in a space overhead.

The storage overhead occurring in both the RPS method and the DDC method is removed by the *space-efficient data cubes* (SEDC) method [19]. The SEDC method provides two strategies to construct the data cube, called the *space-efficient relative prefix sum* (*SRPS*) technique and the *space-efficient dynamic data cubes* (*SDDC*) technique. The SRPS technique requires the query cost and update cost that are less than or equal to those of the RPS method; at the same time, the SDDC approach needs the query cost and update cost that are less than or equal to those of the DDC method.

The above methods attempt to handle data cubes of any dimensionality by dealing with all the dimensions simultaneously and treating the different dimensions uniformly. These algorithms are typically complex, and it is difficult to prove their correctness and analyze their performance. The *iterative data cube* (*IDC*) method [16, 18] was proposed to trade off the query cost and the update cost. A different method can be applied to each dimension. Different one-dimensional techniques can be combined, resulting in a diverse variety of IDC frameworks. Therefore, the IDC technique can be used easily for cube construction and does not suffer from the weakness of producing space overhead for dense data cubes.

In 2001, Chun *et al.* [3] proposed an additive R-tree-like index structure called the Δ-*tree*, designed to reduce significantly the cost for updating the data cube. However, this method requires extra space overhead. Recently, Wang *et al.* [22] have presented a *condensed cube* that reduces the size of data cubes. Even though this approach does well with sparse data cubes, for dense date cubes, the condensed cube technique is nothing new but a Group-By method.

Performance analysis establishes that the PS method has the minimum query cost and maximum update cost. In contrast, the original array *A* has the maximum query cost and minimum update cost. Table 1 shows the analysis of other typical methods, whose query costs and update costs are between these two extremes. Note that the *HRC* [2] cube with a height of 2 is called the *local prefix sum* (*LPS*) cube [18].

Table 1. Trade-off between query cost and update cost for one-dimensional techniques

One-dimensional techniques	**Query cost** (worst case)	**Update cost** (worst case)	**Note**
Original array *A*	n	1	
LPS	$\left\lceil \frac{n}{k} \right\rceil + 1$	k	***k***: block size
SDDC	$2\lceil \log_2 n \rceil$	$\lceil \log_2 n \rceil$	
SRPS	4	$2\sqrt{n} - 2$	$\boldsymbol{k} = \sqrt{n}$
PS	2	n	

3 Analysis of the average query cost and update cost for data cubes

Existing approaches have been used to evaluate the worst query or update cases. However, they do not seem to have been used to find suitable data cubes with a specific query (or update) ratio. In this section, we shall analyze the average query cost and update cost to find the suitable data cube with a particular query (or update) ratio. The original array *A*, the LPS method and the PS method will all be analyzed.

The query cost and update cost are evaluated in terms of the number of cells that must be accessed, which is proportional to the number of accessed cells with accessed disk pages. In this paper, we do not care about the CPU time. For simplicity without loss of generality, reading and writing a cell are assumed to have the same unit cost. According to the IDC method [16, 18], if the one-dimensional average query (or update) cost is known, the *d*-dimensional average query (or update) cost can be computed by multiplying the one-dimensional cost.

Analysis of one-dimension

Definition 1 (Average query cost) Let Q be a element of query set, $C_q(n)$ be the total query cost function and *n* be the size of dimension. We define the average query cost function $C_{aq}(n)$ to be $C_q(n) / \sum_{Q=1}^{n} Q$, where $\sum_{Q=1}^{n} Q = \frac{n(n+1)}{2}$.

Definition 2 (Average update cost) Let $C_u(n)$ be the total update cost function and *n* be the size of dimension. We define the average update cost function $C_{au}(n)$ to be $C_u(n) / n$.

Based on Definitions 1 and 2, the average query and update costs of the three data cube construction methods are analyzed as follows:

1. Original array *A*:
   (a) Average query cost: By Definition 1,

   $$C_{aq}(n) = \sum_{Q=1}^{n} Q(n-Q+1) \Big/ \frac{n(n+1)}{2} = \frac{n+2}{3}$$

   (b) Average update cost: $n / n = 1$
2. LPS method ($k = 2$):
   (a) Average query cost:
   The divide-and-conquer approach is taken to analyze the average query cost of the LPS method. The LPS cube is divided into two equal sub-cubes. The total query cost becomes,

   $$C_q(n) = 2C_q(\frac{n}{2}) + (\frac{n}{2})^2\, C_{aq2}(n) \qquad (1)$$

   where $C_{aq2}(n)$ denotes the average query cost of the query set (C_{q2}) whose query ranges both sub-cubes. Every query in C_{q2} can be divided into two sub-queries. Fig. 4 shows a query of C_{q2}, $n = 8$. The query Sum(*A*[1] : *A*[5]) can be divided into two sub-queries Sum(*A*[3] : *A*[1]) and Sum(*A*[4] : *A*[5]).

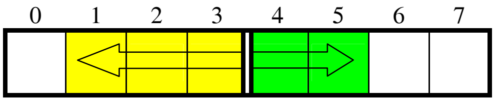


Fig. 4. n = 8. The query Sum(A[1] : A[5]) can be divided into two sub-queries Sum(A[3] : A[1]) and Sum(A[4] : A[5])

Thus, the average query cost of C_{q2} is the sum of the average cost of the left sub-query and the right sub-query.

$$C_{aq2}(n) = C_{aqRL}(\frac{n}{2}) + C_{aqLR}(\frac{n}{2}) \tag{2}$$

$$C_{aqLR}(n) = 2\sum_{Q=1}^{n/2} Q \;/\; n = \frac{n+2}{4}$$

$$C_{aqRL}(n) = (2\sum_{Q=1}^{n/2} Q + \frac{n}{2}) \;/\; n = \frac{n+4}{4}$$

$$C_{aqLR}(n) + C_{aqRL}(n) = \frac{n+2}{4} + \frac{n+4}{4} = \frac{n+3}{2} \tag{3}$$

Introducing Eq.(3) into Eq.(2), we get $C_{aq2}(n)$

$$C_{aq2}(n) = C_{aqLR}(\frac{n}{2}) + C_{aqRL}(\frac{n}{2}) = \frac{n/2+3}{2} \tag{4}$$

Introducing Eq.(4) into Eq.(1), we get the recursive function of total query cost as follows:

$$C_q(n) = 2C_q(\frac{n}{2}) + (\frac{n}{2})^2 (\, C_{aqLR}(\frac{n}{2}) + C_{aqRL}(\frac{n}{2}) \,)$$

$$= \ldots$$

$$= 2^h C_q(\frac{n}{2^h}) + \frac{n^3}{2^2}(\frac{1}{2^{2h}} + \ldots + \frac{1}{2^4} + \frac{1}{2^2}) + \frac{3n^2}{2^2}(\frac{1}{2^h} + \ldots + \frac{1}{2^2} + \frac{1}{2^1})$$

Let $n = 2^h$, and $C_q(1) = 1$,

$$C_q(n) = \frac{n^3}{12} + \frac{3n^2}{4} + \frac{n}{6} \tag{5}$$

Substituting Eq.(5), we get the average query cost:

$$C_{aq}(n) = C_q(n) \;/\; \frac{n(n+1)}{2} = \frac{3}{2} + \frac{n-1}{6} - \frac{1}{n+1} \tag{6}$$

(b) Average update cost: $\frac{3}{2}n \;/\; n = \frac{3}{2}$

3. PS method:

(a) Average query cost: $(2\sum_{Q=1}^{n} Q - n) \;/\; \frac{n(n+1)}{2} = \frac{2n}{n+1}$

(b) Average update cost: $\sum_{Q=1}^{n} Q \;/\; n = \frac{n+1}{2}$

Table 2 shows the result of the analysis of the original array *A*, the LPS method and the PS method for *d*-dimension. Assume that the query ratio is *q*, and the update ratio is u = (1 - *q*). If the total number of queries and updates is *N*, then the total cost $C = N \times q \times C_{aq}(n) + N \times u \times C_{au}(n)$. The most suitable cube varies for different query (or update) ratios. For example, these data cubes in Fig. 5 are two-dimensional, where *N* = 10,000 and the size of dimension is 32. A higher query ratio corresponds to a better PS cube, and a lower query ratio indicates a superior LPS cube and also a better original array *A*.

Table 2. Average query cost and update cost of the three data cubes

Data cube	Average query cost	Average update cost
Original array *A*	$(\frac{n+2}{3})^d$	1
LPS (*k* = 2)	$(\frac{3}{2}+\frac{n-1}{6}-\frac{1}{n+1})^d$	$(\frac{3}{2})^d$
PS	$(\frac{2n}{n+1})^d$	$(\frac{n+1}{2})^d$

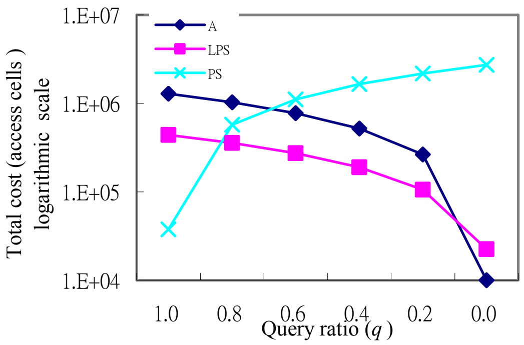


Fig. 5. The total query and update costs of data cubes for *d* = 2, *N* = 10,000, *n* = 32

4 Flexible Data Cube (FDC)

According to the query (or update) ratio, exponential time is required to find the most appropriate pre-aggregated data cube. For example, *n*! data cubes exist in an one-dimension situation, and (n^d)! data cubes exist in a *d*-dimension situation. The Flexible Data Cube (FDC) method that we shall propose right here is a heuristic method to select or integrate any two pre-aggregation techniques for each dimension. The most suitable FDC data cube can be obtained in linear time. Here, we shall use the original array *A*, LPS, and PS to construct the FDC data cube.

Consider the one-dimensional FDC method:

1. Array A is the initial data cube.
2. Array A is divided into two parts (sub-cubes). The first sub-cube has the smaller index value of k', the other k'' ($= n - k'$) cells are in second sub-cube. Initially, $k' = 0$.
3. Any two methods can be selected for the two sub-cubes, and the minimum average query cost and update cost of the data cube FDC_m can be maintained.
4. $k' = k'+1$. Repeat Step 3 until $k' = n - 1$.
5. The most suitable (optimal) FDC data cube (FDC_{opt}) is the one with the minimum FDC_m generated in Step 3.

When $k' = 0$, the array A, LPS, and PS are all special cases of FDC cubes. Finding the FDC_{opt} cube requires only linear time ($\leq 3\times3\times n = 9n$). In multiple dimension cases, one-dimensional techniques are applied in each dimension [16, 18]. The analysis of the FDC average query cost and update cost is similar to the LPS method mentioned in the previous section.

Analysis of one-dimension case

Based on Definition 1 and 2, the average query cost and update cost of one of the FDC cubes are as follows:

1. Average query cost:

   $C_{aqFDC}(n)$ = (total cost for querying the first sub-cube + total cost for querying the second sub-cube + total cost for querying the two sub-cubes simultaneously) / total number of queries. Therefore, $C_{aqFDC}(n)$ =

   $$\left(\frac{k'(k'+1)}{2}C_{aqi}(k') + \frac{k''(k''+1)}{2}C_{aqj}(k'') + k'k''(C_{aqRLi}(k') + C_{aqLRj}(k''))\right) / \frac{n(n+1)}{2},$$

   where $i, j \in \{A, \text{LPS}, \text{PS}\}$, and $C_{aqi}(k')$ denotes the average query cost of the first sub-cube which follows the i method. Similarly, $C_{aqj}(k'')$ denotes the average query cost of the second sub-cube which follows the j method. The item $(C_{aqRLi}(k') + C_{aqLRj}(k''))$ denotes the average query cost of the query set whose query ranges covers both of the sub-cubes.

2. Average update cost:

   $C_{auFDC}(n)$ = (total cost for updating the first cube + total cost for updating the second cube) / total number of updates. Therefore,

   $C_{auFDC}(n) = C_{ui}(k') + C_{uj}(k'') / n$, where $i, j \in$ {A, LPS, PS}, and $C_{ui}(k')$ denotes the total update cost of the first sub-cube which applies the i method. Similarly, $C_{uj}(k'')$ denotes the total update cost of the second sub-cube which applies the j method.

According to Table 2 and the above analysis, the average query cost and update cost of any FDC can be obtained. For one of the one-dimensional FDC cubes, n = 8 and k' = 4. The FDC cube's first part is an LPS sub-cube and second part is a PS sub-cube. The average query cost and update cost are as follows.

1. $C_{aqFDC}(8) = \frac{4\times5}{2}\times\frac{9}{5} + \frac{4\times5}{2}\times\frac{8}{5} + 4\times4(3+\frac{15}{8}) \;/\; \frac{8\times9}{2} = \frac{28}{9}$

2. $C_{auFDC}(8) = (4\times\frac{3}{2} + 4\times\frac{9}{2})/8 = 3$

$$\text{FDC}_{opt} = min\{q\times C_{aqFDC} + u\times C_{auFDC}\} \qquad (7)$$

In Eq.(7), the cost of FDC$_{opt}$ is the minimum among the FDC cubes. To analyze a high-dimensional data cube, according to the IDC method [16, 18], once the one-dimensional average cost is figured out, the d-dimensional average cost can be computed by multiplying the one-dimensional cost.

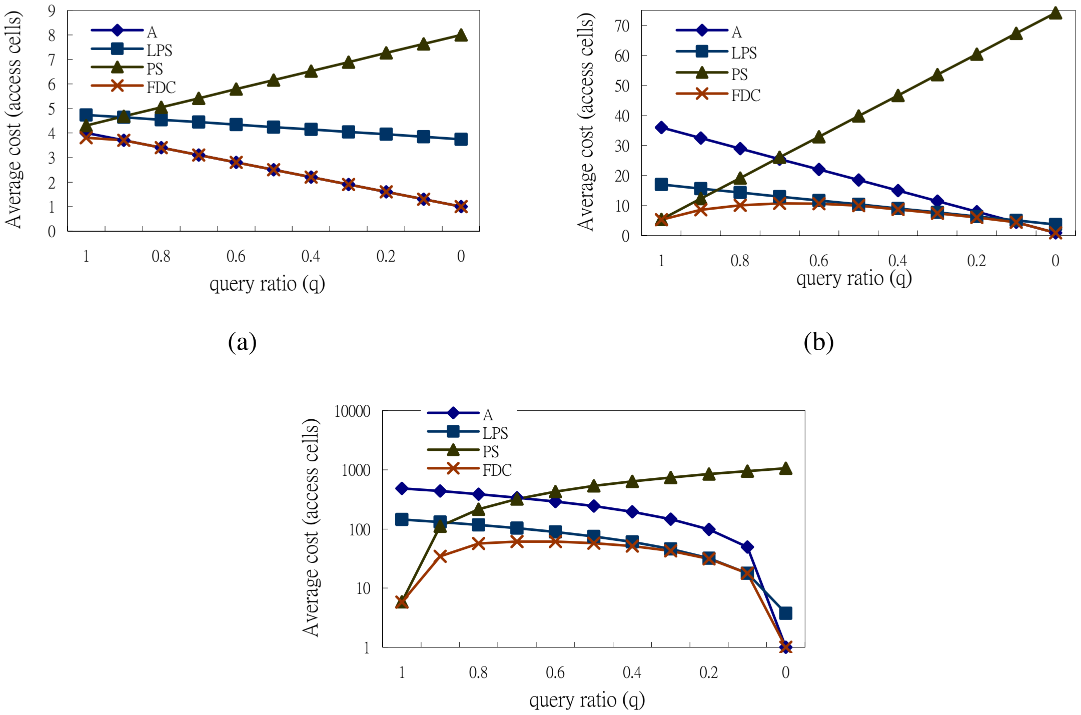
5 Performance Analysis

In this section, we shall compare the performance of our FDC$_{opt}$ with those of the original array A, the LPS method, and the PS method. Without loss of generality, assume every different query range has the same probability of being queried, and every cell has the same probability of getting updated. For simplicity, consider a data cube that has the same size in each dimension. In this case, the average cost equals $q\times C_{aq}(n) + u\times C_{au}(n)$.

In Fig. 5, the best data cube differs from situation to situation (with different query ratios). Fig. 6 shows how the methods compare on the average cost in the two-dimension case at different query (or update) ratios. The X axis is query ratio and the Y axis is average query cost. While n = 4,16 or 64, the average cost of the FDC$_{opt}$ is less than or equal to the other techniques in different query ratios (from 1 to 0). When the query ratio is q = 0, the original array A is a special case of the FDC$_{opt}$. When q = 1, the PS cube is a special case of the FDC$_{opt}$. In these three cases, the larger the size of each dimension, the greater the difference between the FDC$_{opt}$ and the other methods. Note that the Fig. 6(c) uses logarithmic scale for Y axis.

Fig. 7 shows how the methods compare on the average cost in the four-dimension case with different dimensionalities. The X axis is the size of dimension and the Y axis is average query cost. Fig. 7 uses logarithmic scales for both X and

Y axes. While q = 1, 0.9, 0.1 or 0, the FDC$_{opt}$ is less than or equal to the other techniques for various dimension sizes. Fig. 7(a) shows that the PS cube is a special case of the FDC$_{opt}$. When q = 1, the performances of the FDC$_{opt}$ and the original array A are in agreement. In Fig. 7(b), the FDC$_{opt}$ outperforms the other methods, where q = 0.9. Fig. 7(c) also reveals that the FDC$_{opt}$ is the best. When n is smaller than or equal to 16, the average costs of FDC$_{opt}$ and LPS are about the same. Fig. 7(d) illustrates that the original array A is a special case of the FDC$_{opt}$, and thus both the average costs of FDC$_{opt}$ and array A are 1.



(a)

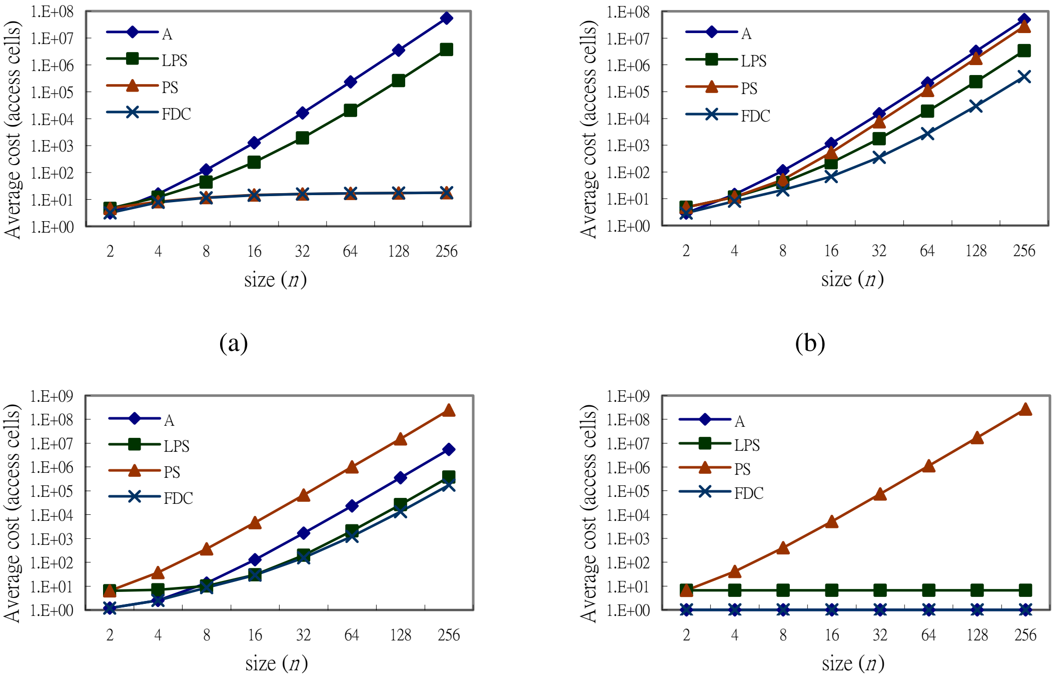
(b)

(c)

Fig. 6. Average query cost and update cost at different query ratios: (a) $d = 2$, $n = 4$ (b) $d = 2$, $n = 16$ (c) $d = 2$, $n = 64$

6 Conclusions

Range-sum queries on data cubes constitute an important tool for analysis. Techniques have been proposed to accelerate range-sum queries by pre-aggregating some specified cells in data cubes. In this paper, we have proposed a novel approach that takes both the query cost and the update cost into consideration to construct the spatially optimal data cube. In addition, we have also proposed a new method called the FDC method. It is a flexible and optimal framework that supports a model to select or integrate pre-aggregating techniques for each dimension. The FDC method outperforms other methods at any query (or update) ratio, and it requires only linear time to determine the best FDC. Our analysis has revealed that the FDC method does provide an effective and efficient framework for pre-aggregating data cubes. In the future, the authors plan to develop new techniques that can efficiently support sparse data sets.



(a)

(b)

(c)

(d)

Fig. 7. Average query cost and update cost for different dimension sizes: (a) $d = 4$, $q = 1$ (b) $d = 4$, $q = 0.9$ (c) $d = 4$, $q = 0.1$ (d) $d = 4$, $q = 0$

References

[1] R. Agrawal, A. Gupta, and S. Sarawagi. Modeling Multidimensional Databases. In *Proc. of the 13th Intl. Conf. on Data Engineering*, pp. 232-243, Apr. 1997.

[2] C.Y. Chan and Y.E. Ioannidis. Hierarchical Cubes for Range-Sum Queries. In *Proc. of the 25th Intl. Conf. on Very Large Data Bases*, pp. 675-686, Sep. 1999.

[3] S.-J. Chun, C.-W. Chung, J.-H. Lee, and S.-L. Lee. Dynamic Update Cube for Range-Sum Queries. In *Proc. of the 27th Intl. Conf. on Very Large Data Bases*, pp. 521-530, Sep. 2001.

[4] E. F. Codd, S. B. Codd, and C. T. Salley. Providing OLAP (On-Line Analytical Processing) to User-Analysts: An IT Mandate. *Technical Report*, 1993.

[5] S. Geffer, D. Agrawal, and A. El Abbadi. The Dynamic Data Cube. In *Proc. of the Intl. Conf. on Extending Database Technology*, pp. 237-253, Mar. 2000.

[6] S. Geffer, D. Agrawal, A. El Abbadi, and T. Smith. Relative Prefix Sums: An Efficient Approach for Querying Dynamic OLAP Data Cubes. In *Proc. of the 15th Intl. Conf. on Data Engineering*, pp. 328-335, Mar. 1999.

[7] S. Geffer, M. Riedewald, D. Agrawal, and A. El Abbadi. Data Cube in Dynamic Environments. *IEEE Data Engineering Bulletin*, Vol. 22, No. 4, pp. 31-40, 1999.

[8] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss. Optimal and Approximate Computation of Summary Statistics for Range Aggregates. In *Proc. of the 20th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pp. 227-236, May 2001.

[9] 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 and Knowledge Discovery,* Vol. 1, No. 1, pp. 29-53, 1997.

[10] D. Gunopulos, G. Kollios, V. J. Tsotras, and C. Domeniconi. Approximating Multi-Dimensional Aggregate Range Queries over Real Attributes. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pp. 463-474, May 2000.

[11] C. T. Ho, R. Agrawal, N. Megiddo, and R. Srikant. Range Queries in OLAP Data Cubes. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pp. 77-88, June 1997.

[12] I. Lazaridis and S. Mehrotra. Progressive approximate Aggregate Queries with a Multi-Resolution Tree Structure. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pp. 401-412, May 2001.

[13] W. Liang, H. Wang, and M. E. Orlowska. Range Queries in Dynamic OLAP Data Cubes. *Data & Knowledge Engineering*, Vol. 34, No. 1, pp. 21-38, 2000.

[14] The OLAP Council. MD-API the OLAP Application Program Interface Version 2.0 Specification, 1998.

[15] V. Poosala and V. Ganti. Fast Approximate Answers to Aggregate Queries on a Data Cube. In *Proc. of the 11th Intl. Conf. on Scientific and Statistical Database Management*, pp. 24-33, Aug. 1999.

[16] M. Riedewald, D. Agrawal, and A. E. Abbadi. The Iterative Data Cube. *University of California, Santa Barbara, Computer Science Technical Report,* 2000.

[17] M. Riedewald, D. Agrawal, and A. E. Abbadi. pCube: Update-Efficient Online Aggregation with Progressive Feedback. In *Proc. of the 12th Intl. Conf. on Scientific and Statistical Database Management*, pp. 95-108, July 2000.

[18] M. Riedewald, D. Agrawal, and A. E. Abbadi. Flexible Data Cubes for Online Aggregation. In *Proc. of the Intl. Conf. on Database Theory,* pp. 159-173, Jan. 2001.

[19] M. Riedewald, D. Agrawal, A. E. Abbadi, and R. Pajarola. Space-Efficient Data Cubes for Dynamic Environments. In *Proc. of the Intl. Conf. on Data Warehousing and Knowledge Discovery,* pp. 24-33, Sep. 2000.

[20] R. R. Schmidt and C. Shahabi. How to Evaluate Multiple Range-Sum Queries Progressively. In *Proc. of the 21st ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pp. 133-141, June 2002.

[21] J. Shanmugasundaram, U. Fayyad, and P. Bradley. Compressed Data Cubes for OLAP Aggregate Query Approximation on Continuous Dimensions. In *Proc. of the 15th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining,* pp. 223-232, Aug. 2000.

[22] W. Wang, H. Lu, J. Feng, and J. X. Yu. Condensed Cube: An Effective Approach to Reducing Data Cube Size. In *Proc. of the 18th Intl. Conf. on Data Engineering*, pp155-165, Apr. 2002.

[23] Y.-L. Wu, D. Agrawal, and A. E. Abbadi. Using Wavelet Decomposition to Support Progressive and Approximate Range-Sum Queries over Data Cubes. In *Proc. of the 9th Intl. Conf. on Information and Knowledge Management*, pp. 414-421, Nov. 2000.