Complex Aggregation at Multiple Granularities

Kenneth A. Ross[1], Divesh Srivastava[2], Damianos Chatziantoniou[3]

[1] Columbia University, New York, NY 10027, USA
[2] AT&T Labs–Research, Florham Park, NJ 07932, USA
[3] Stevens Institute of Technology, Hoboken, NJ 07030, USA

Abstract. Datacube queries compute simple aggregates at multiple granularities. In this paper we examine the more general and useful problem of computing a complex subquery involving multiple dependent aggregates at multiple granularities. We call such queries "multi-feature cubes." An example is "Broken down by all combinations of month and customer, find the fraction of the total sales in 1996 of a particular item due to suppliers supplying within 10% of the minimum price (within the group), showing all subtotals across each dimension." We classify multi-feature cubes based on the extent to which fine granularity results can be used to compute coarse granularity results; this classification includes distributive, algebraic and holistic multi-feature cubes. We provide syntactic sufficient conditions to determine when a multi-feature cube is either distributive or algebraic. This distinction is important because, as we show, existing datacube evaluation algorithms can be used to compute multi-feature cubes that are distributive or algebraic, without any increase in I/O complexity. We evaluate the CPU performance of computing multi-feature cubes using the datacube evaluation algorithm of Ross and Srivastava. Using a variety of synthetic, benchmark and real-world data sets, we demonstrate that the CPU cost of evaluating distributive multi-feature cubes is comparable to that of evaluating simple datacubes. We also show that a variety of holistic multi-feature cubes can be evaluated with a manageable overhead compared to the distributive case.

1 Introduction

Decision support systems aim to provide answers to complex queries posed over very large databases. The databases may represent business information (such as transaction data), medical information (such as patient treatments and outcomes), or scientific data (such as large sets of experimental measurements). The vast quantities of data contain enough information to answer questions of importance to the application user. Useful queries in the domains above include:

- Broken down by supplier, month and item, find the total sales in 1996, including all subtotals across each dimension.
- Broken down by hospital, diagnosis and treatment, find the average life expectancy, including all subtotals across each dimension.

The electronic mail addresses of the authors are kar@cs.columbia.edu, divesh@research.att.com and damianos@cs.stevens-tech.edu.

Each of these queries is an example of a datacube query [GBLP96]. Datacube queries allow one to compute aggregates of the data at a variety of granularities. The first query above would generate aggregate data at eight different granularities including total sales by (a) supplier, (b) month, (c) item, (d) supplier and month, (e) supplier and item, (f) month and item, (g) supplier, month and item, and (h) the empty set (i.e., the total overall sales). A datacube could be computed by separately computing the aggregate at each granularity. However, it is also possible to compute aggregates at several coarser levels of granularity at the same time as computing aggregates at finer levels of granularity. Such algorithms are presented in [GBLP96, AAD$^+$96, ZDN97, RS97].

A different kind of decision support query has been considered in [CR96], involving aggregation queries in which multiple dependent aggregates are computed within each group. An example of such a query is the following:

Q0: For each item, find its minimum price in 1996, and the total sales among all minimum price tuples.

[CR96] presented an extended SQL syntax that allows a succinct representation of such queries. An experimental study demonstrated that such queries can be efficiently evaluated. In contrast, standard SQL representations of the same queries are verbose and redundant, leading to queries that are hard to understand, to maintain, and to optimize.

In this paper, we consider complex decision support queries in which multiple dependent aggregates are computed at a variety of granularities. In particular, we would like to be able to ask queries like that of Query Q0 above, but replacing the phrase "For each item" by "Grouping by all subsets of {supplier, month, item}." We call such queries *multi-feature cubes*, and illustrate the practical utility of such queries using a number of examples. The main contributions of this paper are the following.

Classification (Section 4) We classify multi-feature cubes based on their *degree of incrementality*. We extend the notions of *distributive*, *algebraic*, and *holistic* aggregates from [GBLP96] to our more general context.

Identification (Section 5) We provide syntactic sufficient conditions on multi-feature cube queries to determine when they are distributive or algebraic. The evaluation of such queries can be performed particularly efficiently, so it is important to be able to identify them syntactically. These conditions admit a large class of multi-feature cube queries beyond those expressible as simple datacubes.

Evaluation (Section 6) We present an algorithm that incrementally computes the coarser granularity output of a distributive multi-feature cube using the finer granularity output of the multi-feature cube. We show that this algorithm can be used in conjunction with previously proposed techniques for efficiently evaluating datacubes to evaluate multi-feature cubes that are distributive or algebraic, with the same I/O complexity. We also discuss the suitability of previously proposed datacube evaluation techniques for efficiently evaluating holistic multi-feature cubes.

Performance (Section 7) We evaluate the CPU performance of computing multi-feature cubes using the datacube evaluation algorithm of Ross and Srivastava [RS97]. Using a variety of synthetic, benchmark and real-world data sets, we demonstrate that the CPU cost of evaluating distributive multi-feature cubes is comparable to that of evaluating simple datacubes. We also show that a variety of holistic multi-feature cubes can be evaluated with a manageable overhead compared to the distributive case.

We can hence ask considerably more sophisticated queries than datacube queries without incurring a significant cost increase! All the examples in this paper will use the relation `SUPPLIES(Supplier, Customer, Item, Year, Month, Day, Price, Sales, Delay)` from a business application database. Suppliers supply items to customers. The unit price and sales (in dollars) of items ordered by the customer from the supplier on the given date are stored in `Price` and `Sales`. Orders placed on that date are delivered after `Delay` days.

2 Background

2.1 The Datacube: Aggregation at Multiple Granularities

In [GBLP96], Gray et al. present the datacube, which allows the computation of aggregates of the data at multiple granularities. We refer the reader to [GBLP96] for the syntax and semantics of such queries in general, and present an example below to aid intuition.

Example 2.1: Suppose that we want to ask the following datacube query: grouping by all subsets of {`Supplier, Customer, Item, Month`}, find the total sales among all tuples from 1996. One could write this query as:

```
SELECT   Supplier, Customer, Item, Month, SUM(Sales)
FROM     SUPPLIES
WHERE    Year = 1996
CUBE BY  Supplier, Customer, Item, Month
```

The meaning of this datacube is the union of the results of 16 SQL queries, obtained as follows: for each subset $\tilde{\mathtt{B}}$ of the `CUBE BY` attributes, the `CUBE BY` clause is replaced by `GROUP BY` $\tilde{\mathtt{B}}$, and any attribute in the `SELECT` clause not in $\tilde{\mathtt{B}}$ is replaced by the special constant value `ALL`.[4] □

2.2 Querying Multiple Features of Groups

In [CR96], Chatziantoniou and Ross present an extension of SQL that allows one to query multiple features of groups in relational databases. We refer the reader to [CR96] for the syntax and semantics of such queries, and present an example below to aid intuition.

[4] Recall that in standard SQL only attributes in the `GROUP BY` clause, aggregates and constant values can appear in the `SELECT` clause.

Example 2.2: Suppose that we want to ask the following query: for each customer, for each item, and for each month in 1996, find the total sales among all minimum price suppliers of that item for that month. In the SQL extension of [CR96], one could write this query as:

```
SELECT    Customer, Item, Month, SUM(R.Sales)
FROM      SUPPLIES
WHERE     Year = 1996
GROUP BY  Customer, Item, Month : R
SUCH THAT R.Price = MIN(Price)
```

The meaning of this query can be understood as follows. First, all tuples in the `SUPPLIES` relation that satisfy the condition "`Year = 1996`" are selected, and these tuples are grouped based on their values of the grouping attributes `Customer, Item` and `Month` into multiple groups, say $g_1, \ldots, g_r$. For each group of tuples g_i, the minimum price m_{g_i} among the tuples of g_i is computed, and grouping variable `R` ranges over all tuples in group g_i whose price is equal to m_{g_i}. The sum of sales of the tuples in g_i that `R` ranges over is then computed, and associated with the values of the grouping attributes of g_i in the query result. □

3 Multi-Feature Cubes

The work of Chatziantoniou and Ross from Section 2.2 motivates us to try to ask more general queries at multiple granularities. Consider Example 2.2 once more, and suppose that we wish to ask the same query at different time granularities. For example, finding minimum price suppliers over the whole year, and summing their sales, is commonly called a "roll-up" query. If we wish to find the minimum price suppliers on each day, and sum their sales, then we would be "drilling down" to a finer granularity. If we wish to answer this query at all possible granularities within a given set of grouping variables, then we are performing an operation analogous to the datacube, which we call a *multi-feature cube*.

Example 3.1: We shall use the following queries throughout this paper:

Q1: Grouping by all subsets of {`Supplier, Customer, Item, Month`} find the minimum price among all tuples from 1996, and the total sales among all such minimum price tuples.

Q2: Grouping by all subsets of {`Supplier, Customer, Item, Month`} find the minimum price among all tuples from 1996, and the fraction of the total sales due to tuples whose delay is less than 10 days and whose price is within 25%, within 50% and within 75% of the minimum price.

Q3: Grouping by all subsets of {`Supplier, Customer, Item, Month`} find the minimum price among all tuples from 1996, the maximum and minimum delays within the set of all minimum price tuples, and the fraction of the total sales due to tuples that have maximum delay within the set of all minimum price tuples, and the fraction of the total sales due to tuples that have minimum delay within the set of all minimum price tuples. □

```
Q:  SELECT     B₁, ..., Bₖ, f₁(A₁), ..., fₙ(Aₙ)
    FROM       T₁, ..., Tₚ
    WHERE      Cond
    CUBE BY    B₁, ..., Bₖ : R₁, ..., Rₘ
    SUCH THAT  S₁ AND ... AND Sₘ
```

Fig. 1. Syntax for Multi-Feature Cube Queries

Just as the multi-feature queries of [CR96] can be expressed using standard features of SQL such as views and/or subqueries, multi-feature cubes can also be expressed using the datacube and views/subqueries. However, as argued in [CR96], the resulting expressions are both complex and repetitious, leading to queries that are difficult to understand, maintain, and optimize, Thus, we prefer to extend the succinct syntax of [CR96] with the `CUBE BY` clause of [GBLP96].

3.1 A Combined Syntax for Multi-Feature Cubes

A *multi-feature cube* query Q has the syntax described in Figure 1. The `FROM` and the `WHERE` clauses in the multi-feature cube are identical to the corresponding clauses in the syntactic extensions of [CR96] and [GBLP96], which are unchanged from standard SQL. The `CUBE BY` clause in the multi-feature cube combines the `CUBE BY` clause from [GBLP96] with the specification of grouping variables $R_1, \ldots, R_m$ of the `GROUP BY` clause from [CR96]. The `SELECT` and the `SUCH THAT` clauses in the multi-feature cube are identical to the corresponding clauses in the syntactic extension of [CR96]. The meaning of the multi-feature cube is the union of the results of all 2^k queries of the form:

```
SELECT     B₁, ..., Bₖ, f₁(A₁), ..., fₙ(Aₙ)
FROM       T₁, ..., Tₚ
WHERE      Cond
GROUP BY   B̃ : R₁, ..., Rₘ
SUCH THAT  S₁ AND ... AND Sₘ
```

where $\tilde{B}$ is an arbitrary subset of $\{B_1, \ldots, B_k\}$, and any B_i not in $\tilde{B}$ that appears in the `SELECT` clause is evaluated as the special constant value `ALL`.

When we require that grouping variable R_j ranges over a *subset* of the tuples that grouping variable R_i (where $i < j$) ranges over, we simply write "R_j `in` R_i" in the condition S_j of the `SUCH THAT` clause. This notation is a convenient shorthand for a query in which the conditions in the `SUCH THAT` clause for S_i are repeated in S_j (for R_j rather than R_i).

Example 3.2: We now express the queries of Example 3.1 using our syntax.

```
Q1: SELECT     Supplier, Customer, Item, Month, MIN(Price), SUM(R.Sales)
    FROM       SUPPLIES
    WHERE      Year = 1996
    CUBE BY    Supplier, Customer, Item, Month : R
    SUCH THAT  R.Price = MIN(Price)
```

```
Q2: SELECT     Supplier, Customer, Item, Month, MIN(Price),
               SUM(R1.Sales), SUM(R2.Sales), SUM(R3.Sales), SUM(Sales)
    FROM       SUPPLIES
    WHERE      Year = 1996
    CUBE BY    Supplier, Customer, Item, Month : R1, R2, R3
    SUCH THAT  R1.Price <= 1.25*MIN(Price) AND R1.Delay < 10
               AND R2.Price <= 1.50*MIN(Price) AND R2.Delay < 10
               AND R3.Price <= 1.75*MIN(Price) AND R3.Delay < 10

Q3: SELECT     Supplier, Customer, Item, Month, MIN(Price), MIN(R1.Delay),
               MAX(R1.Delay), SUM(R1.Sales), SUM(R2.Sales), SUM(R3.Sales)
    FROM       SUPPLIES
    WHERE      Year = 1996
    CUBE BY    Supplier, Customer, Item, Month : R1, R2, R3
    SUCH THAT  R1.Price = MIN(Price) AND R2 in R1 AND R2.Delay =
               MIN(R1.Delay) AND R3 in R1 AND R3.Delay = MAX(R1.Delay)
```

4 Classifying Multi-Feature Cubes

To better understand issues arising in the evaluation of multi-feature cubes, we propose a classification based on the notion of incremental evaluability.

Definition 4.1: (Group, Granularity) Let Q be a multi-feature cube query on a database D, and let $\mathtt{B}_1, \ldots, \mathtt{B}_k$ be the attributes mentioned in the `CUBE BY` clause of Q. Each instance v of attributes $\mathtt{B}_1, \ldots, \mathtt{B}_k$ (including instances involving the special `ALL` value) is called a *group*.

Groups v and v' are said to be *at the same level* if they take the value `ALL` on exactly the same attributes. The set of all groups at the same level is called a *granularity*, and is denoted by its set of non-`ALL` attributes. Group v' is *coarser than* v (or, v is *finer than* v') if $v' \neq v$ and every non-`ALL` value of a (grouping) attribute in v' is also the value of that attribute in v. Also, the granularity of such a group v' is said to be *coarser than* the granularity of group v. □

Datacubes and multi-feature cubes can be evaluated using multiple passes over the base data. While such an evaluation technique may be required in general, a large number of datacubes (those using distributive aggregate functions, in the terminology of [GBLP96]) can be evaluated much more efficiently, by *incrementally* computing the output of the datacube at a coarser granularity using only the output of the datacube at a finer granularity. We capture this property in our definition of *distributive* multi-feature cubes.

Definition 4.2: (Distributive Multi-Feature Cube) Consider the multi-feature cube Q given by the syntax of Figure 1. Let $\tilde{\mathtt{B}}_1$ and $\tilde{\mathtt{B}}_2$ denote arbitrary subsets of the `CUBE BY` attributes $\{\mathtt{B}_1, \ldots, \mathtt{B}_k\}$, such that $\tilde{\mathtt{B}}_1$ is a subset of $\tilde{\mathtt{B}}_2$. Let $Q_i,\ i \in \{1, 2\}$ denote the query:

```
SELECT     B1, ..., Bk, f1(A1), ..., fn(An)
FROM       T1, ..., Tp
```

```
WHERE      Cond
GROUP BY   B̃ᵢ : R₁, ..., Rₘ
SUCH THAT  S₁ AND ... AND Sₘ
```

where any $\mathtt{B}_j$ not in $\tilde{\mathtt{B}}_i$ appearing in the `SELECT` clause is evaluated as the constant value `ALL`.

Query Q is said to be a *distributive* multi-feature cube if there is a computable function F such that for all databases D and all Q_1 and Q_2 as above, $output(Q_1, D)$ can be computed via F as $F(output(Q_2, D))$. □

Proposition 4.1: Datacubes that use only distributive aggregate functions are distributive multi-feature cubes. □

Example 4.1: We show that Query Q1 is a distributive multi-feature cube. Suppose that we have computed the aggregates for the granularity `Supplier, Customer, Item, Month` and have kept both `MIN(Price)` and `SUM(R.Sales)` for each group. We now wish to compute the aggregates for the granularity `Supplier, Customer, Item`. We can combine the twelve pairs of values (one per month) into an annual pair of values, as follows: (a) Compute the minimum of the monthly `MIN(Price)` values. This is the annual `MIN(Price)` value. (b) Add up the `SUM(R.Sales)` for those months whose monthly `MIN(Price)` value is equal to the annual `MIN(Price)` value. This is the annual `SUM(R.Sales)` value. □

However, multi-feature cubes may be non-distributive even when *each* aggregate function in the `SELECT` and `SUCH THAT` clauses is distributive. Query Q2 from Example 3.2 is such a non-distributive cube, as the following example illustrates.

Example 4.2: Consider Query Q2, which determines the fraction of the total sales due to tuples whose delay is less than 10 days and whose price is within 25%, within 50% and within 75% of the minimum price.

Suppose that we've computed these aggregates for the granularity `Supplier, Customer, Item, Month` and have kept all of `MIN(Price)`, `SUM(R1.Sales)`, `SUM(R2.Sales)`, `SUM(R3.Sales)` and `SUM(Sales)` for each group. We now wish to compute the aggregates for the granularity `Supplier, Customer, Item`. Unfortunately, we cannot simply combine the twelve tuples of values (one per month) into a global tuple of values. Suppose that (for some group) the minimum price over the whole year is \$110, but that the minimum price for January is \$120. Then we do not know how to combine January's `SUM(R2.Sales)` of \$1000 into the yearly `SUM(R2.Sales)` since we do not know what fraction of the \$1000 came from tuples with price at most \$165; the figure of \$1000 includes contributions from tuples with price up to \$180. □

While not all multi-feature cubes are distributive, it is sometimes possible to "extend" multi-feature cubes by adding aggregates to the `SELECT` clause, such that the modified multi-feature cube *is* distributive. For example, a datacube that has `AVG(Sales)`, but neither `COUNT(Sales)` nor `SUM(Sales)`, in its `SELECT` clause, can be extended to a distributive multi-feature cube by adding `SUM(Sales)` to the `SELECT` clause. The average sales at coarser granularities can now be computed from the average sales and the total sales at finer granularities.

Definition 4.3: (Algebraic and Holistic Multi-Feature Cubes) Consider a multi-feature cube Q. Q is said to be an *algebraic* multi-feature cube if there exists a Q' obtained by adding aggregates to the `SELECT` clause, such that Q' is distributive. Otherwise, Q is said to be a *holistic* multi-feature cube. □

Query Q2 from Examples 3.2 and 4.2 above, is an example of a holistic multi-feature cube since no extension of Query Q2 would be distributive.

5 Identifying Distributive Multi-Feature Cubes

In this section, we identify natural syntactic conditions on multi-feature cubes for them to be distributive. Our first step is to define a binary relation "$\prec$" on the grouping variables $\mathtt{R}_1, \ldots, \mathtt{R}_m$, based on the pattern of attribute references in the `SUCH THAT` clause.

Definition 5.1: ($\mathtt{R}_i \prec \mathtt{R}_j$) Grouping variable $\mathtt{R}_i$ is said to be $<$ grouping variable $\mathtt{R}_j$ if the condition $\mathtt{S}_j$ (defining $\mathtt{R}_j$) in the `SUCH THAT` clause refers to an attribute of $\mathtt{R}_i$. Define $\prec$ to be the reflexive, transitive closure of $<$. □

Note that the syntactic restrictions on the conditions in the `SUCH THAT` clause of multi-feature cubes, from Section 3.1, guarantee that "$\prec$" is a partial order.

Our next step is to identify the pattern of accesses of different grouping variables, i.e., the relationships between the tuples within a group over which the grouping variables range. The following definition identifies an important access pattern of grouping variables.

Definition 5.2: ($\mathtt{R}_i \sqsubset \mathtt{R}_j$) Grouping variable $\mathtt{R}_i$ is $\sqsubset$ grouping variable $\mathtt{R}_j$, if $\mathtt{R}_j$ always ranges over a subset of the tuples that $\mathtt{R}_i$ ranges over. □

It is important to understand that $\mathtt{R}_i \prec \mathtt{R}_j$ does not necessarily imply that $\mathtt{R}_i \sqsubset \mathtt{R}_j$. However, as we show later, such a condition on grouping variables that are related by the partial order "$\prec$" is essential for a multi-feature cube to be distributive. In general, $\mathtt{R}_i \sqsubset \mathtt{R}_j$ if and only if condition $\mathtt{S}_j$ (parameterized by the attributes of $\mathtt{R}_j$) implies condition $\mathtt{S}_i$ (parameterized by the attributes of $\mathtt{R}_i$). In many cases we can identify that $\mathtt{R}_i \sqsubset \mathtt{R}_j$ simply by checking that $\mathtt{S}_j$ contains the shorthand "$\mathtt{R}_j$ `in` $\mathtt{R}_i$."

The multi-feature cube graph, defined below, captures most of the important relationships between grouping variables, and the nature of the defining conditions in the `SUCH THAT` clause for the grouping variables.

Definition 5.3: (Multi-Feature Cube Graph) Given a multi-feature cube query Q, the *multi-feature cube graph* of Q, denoted by $MFCG(Q)$ is a labeled, directed, acyclic graph defined as follows.

The nodes of $MFCG(Q)$ are the grouping variables $\mathtt{R}_1, \ldots, \mathtt{R}_m$ in the `CUBE BY` clause of Q, along with an additional node $\mathtt{R}_0$.

There is a directed edge $(\mathtt{R}_i, \mathtt{R}_j)$, $i < j$, in $MFCG(Q)$ if $\mathtt{R}_i \sqsubset \mathtt{R}_j$ and there does not exist $\mathtt{R}_k$ different from $\mathtt{R}_i$ and $\mathtt{R}_j$, such that $\mathtt{R}_i \sqsubset \mathtt{R}_k \sqsubset \mathtt{R}_j$. For every node $\mathtt{R}_j, j \neq 0$, such that there is no edge of the form $(\mathtt{R}_i, \mathtt{R}_j)$ in *MFCG*(Q), add an edge $(\mathtt{R}_0, \mathtt{R}_j)$.

If there is an edge $(\mathtt{R}_i, \mathtt{R}_j)$ in *MFCG*(Q), node $\mathtt{R}_i$ is referred to as a *parent* of node $\mathtt{R}_j$. If there is a path from $\mathtt{R}_i$ to $\mathtt{R}_j$, in *MFCG*(Q), node $\mathtt{R}_i$ is referred to as an *ancestor* of node $\mathtt{R}_j$.

Node $\mathtt{R}_0$ is associated with the empty label. Each node $\mathtt{R}_i, i \neq 0$, in *MFCG*(Q) is labeled with those aggregate conditions that appear in $\mathtt{S}_i$, the defining condition for $\mathtt{R}_i$ in the `SUCH THAT` clause of Q, but are not part of the label of an ancestor of $\mathtt{R}_i$ in *MFCG*(Q). □

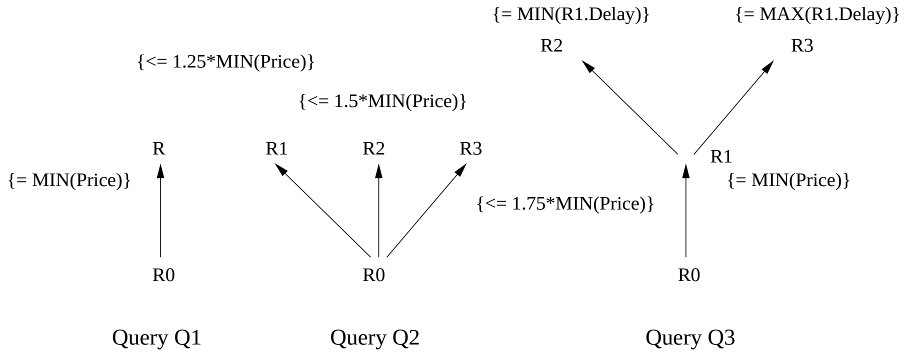

Fig. 2. Example Multi-Feature Cube Graphs

Figure 2 presents the multi-feature cube graphs for Queries Q1, Q2 and Q3. As a final step before identifying conditions for distributivity of multi-feature cubes, we define what it means for a *set* of aggregate functions of attributes to be distributive. This generalizes the characterization, from [GBLP96], of a *single* distributive aggregate function.

Definition 5.4: A set $\tilde{\mathcal{F}}$ of aggregate functions of attributes $\mathtt{f}_1(\mathtt{A}_1)$, ..., $\mathtt{f}_n(\mathtt{A}_n)$ is said to be *distributive*, if (1) each aggregate function $\mathtt{f}_i$ is either distributive or algebraic, and (2) for algebraic aggregate functions $\mathtt{f}_i$, such that $\mathtt{f}_i(\mathtt{A}_i) \in \tilde{\mathcal{F}}$, other aggregate functions of attributes in $\tilde{\mathcal{F}}$ provide the necessary additional information needed for the incremental computation of $\mathtt{f}_i(\mathtt{A}_i)$. □

For example, the set {`MIN(Price), AVG(R.Sales)`} is not distributive, but the set {`MIN(Price), AVG(R.Sales), SUM(R.Sales)`} is distributive.

Theorem 5.1: Consider a multi-feature cube query Q. Let $\tilde{\mathtt{R}}$ denote the set $\{\mathtt{R}_1, \ldots, \mathtt{R}_m\}$ of grouping variables in Q. Query Q is *distributive* if each of the following conditions is satisfied:

C1. The set of $\mathtt{f}_i(\mathtt{A}_i)$'s in the `SELECT` clause of Q is distributive.

C2. For any $\mathtt{R}_i, \mathtt{R}_j \in \tilde{\mathtt{R}}$, $\mathtt{R}_i \prec \mathtt{R}_j$ implies $\mathtt{R}_i \sqsubset \mathtt{R}_j$.

C3. The multi-feature cube graph *MFCG*(Q) is a tree with at most one aggregate condition on each node. Further, the aggregate condition (if any) associated with node $\mathtt{R}_j$ is of one of the forms[5] $\mathtt{R}_j.\mathtt{A}_k = \mathtt{MAX}([\mathtt{R}_i.]\mathtt{A}_k)$, or

[5] $\mathtt{R}_0.\mathtt{A}_k$ is written simply as $\mathtt{A}_k$.

$\mathtt{R}_j.\mathtt{A}_k$ = MIN([$\mathtt{R}_i$.]$\mathtt{A}_k$), where $\mathtt{R}_i$ is the parent of $\mathtt{R}_j$ in *MFCG*(Q).

C4. For each aggregate of the form MAX([$\mathtt{R}_j$.]$\mathtt{A}_i$) (or MIN([$\mathtt{R}_j$.]$\mathtt{A}_i$)) in the SUCH THAT clause of query Q, MAX([$\mathtt{R}_j$.]$\mathtt{A}_i$) (resp. MIN([$\mathtt{R}_j$.]$\mathtt{A}_i$)) appears in the SELECT clause of Q. □

Example 5.1: Consider Queries Q1, Q2 and Q3 from Example 3.2. The first and third queries satisfy the conditions of Theorem 5.1, but the conditions in the SUCH THAT clause of the second query violate Condition C3. □

The conditions of Theorem 5.1 can be easily modified to identify algebraic multi-feature cubes, as shown in the full version of the paper [RSC97].

6 Evaluating Multi-Feature Cubes

When considering the evaluation of multi-feature cube queries, we consider only distributive queries which (in a sense) represent the "fully incremental" queries, and holistic queries which represent the non-incremental queries. Algebraic queries can be evaluated by first transforming them to be distributive.

6.1 The Distributive Case: The *F* Function

The definition of a distributive multi-feature cube (Definition 4.2) requires the existence of a computable function F such that for all databases, the output of the multi-feature cube for a coarser granularity can be computed from the output of the multi-feature cube for finer granularities via F.[6]

When the multi-feature cube is just a simple datacube, the computation of the output for a coarser granularity from the output for a finer granularity is simple for the standard SQL aggregate functions. For example, the value of MAX(A) for a coarser group can be computed by taking the maximum of all the MAX(A) values in its finer groups, and the value of COUNT(A) for a coarser group can be computed as the sum of all the COUNT(A) values in its finer groups. The following example presents the F function for a simple multi-feature cube.

Example 6.1: Consider again Query Q1, which computes the minimum price among all tuples from 1996, and the total sales among all such minimum price tuples, for all subsets of {Supplier, Customer, Item, Month}. The specification of Example 3.2 satisfies the conditions of Theorem 5.1.

We now give the function F to compute coarser granularity results from finer granularity results on this query. Suppose that $\{v_1, \ldots, v_t\}$ are all of the groups at a fine granularity, and v is a coarser group for which we want to generate the output. The aggregates for each group v_j will be a pair of the form (p_j, t_j), where p_j is the minimum price, and t_j is the total sales. We process groups v_j one by one as follows, adjusting the aggregates (p, t) for the coarser group v as we go. F has a local flag f that is initially set to false, indicating that no groups have been processed.

[6] Note that the output of the multi-feature cube at the finest granularity cannot be computed using the function F. This computation can be performed using the techniques suggested in [CR96].

- If f is false then f is set to true, and $(p, t) = (p_j, t_j)$
- else if $(p_j < p)$ then $(p, t) = (p_j, t_j)$
- else if $(p_j = p)$ then $t = t + t_j$
- else do nothing.

At the end of the scan, we return (p, t). □

The multi-feature cube graph of Query Q1 is a simple chain with a single grouping variable, as can be seen from Figure 2. A general algorithm that is applicable whenever the conditions of Theorem 5.1 are satisfied has to deal with many grouping variables and tree-structured multi-feature cube graphs, and is given as `Incremental-Eval` in the full version of the paper [RSC97].

6.2 Algorithms for Distributive Multi-Feature Cubes

Several algorithms have been proposed for computing the datacube [GBLP96, AAD+96, ZDN97, RS97]. The details of these algorithms are not important here. All of these algorithms attempt to compute the datacube by utilizing the lattice structure of the various granularities. Finer granularity results are combined to give results at the next coarser granularity. Some of these algorithms perform optimizations based on the estimated size of answers of groups within the datacube. There are two crucial features of distributive multi-feature cubes that allow the algorithms mentioned above to apply:

1. Coarser granularity aggregates can be computed from finer granularity aggregates by distributivity. In particular, the function F to compute the coarser aggregates can be automatically derived. This F function plays the role that addition would play in evaluating a datacube with aggregate `SUM`.
2. The size of the output for each group is constant (one tuple of fixed size per group) and is easily derived from the query syntax. Thus one can use the same estimation techniques as used in the algorithms above to estimate the size of intermediate results, and employ the same optimization strategies.

As a result of these observations, it becomes clear that each of these algorithms can also be applied to distributive multi-feature cubes *without any increase in I/O complexity* compared with a simple datacube. Since F may be harder to compute than a simple aggregate like `SUM`, there might be an increase in the CPU cost. We address this concern in detail in Section 7.

6.3 The Holistic Case

The algorithms that have been proposed for computing the datacube [GBLP96, AAD+96, ZDN97, RS97] can be divided into two approaches:

- Algorithms that use main memory essentially for storing the input relation, typically in a partitioned and/or sorted fashion [AAD+96, RS97]. Datacube tuples are computed, and immediately flushed to output buffers.
- Algorithms that use main memory essentially for storing the (partially computed) output, typically as a k-dimensional array [GBLP96, ZDN97].

Of these, the algorithms in the first approach are better suited to computing holistic multi-feature cubes. The reason is that the first approach lends itself to the possibility of maintaining all tuples from a group of the input relation simultaneously in memory, allowing for the computation of the holistic multi-feature aggregate over this group. The second approach, on the other hand, maintains partially computed aggregates, obtained by scanning the input relation; such incremental computation cannot be performed for holistic multi-feature cubes.

7 Experimental Evaluation

We used the techniques proposed by Ross and Srivastava [RS97] to implement a variety of multi-feature cube queries. The software used is extensible with respect to the aggregate functions used; arbitrary distributive or non-distributive aggregate functions can be written and linked to the cube-computation code.

For distributive aggregates, the software works as follows: First, combine all tuples that share all `CUBE BY` attributes into a single tuple using the combination function F (which exists for distributive aggregates). After doing so, multiple sorting and scanning steps compute the cube result, as described in [RS97].

For holistic aggregates, there is no initial combining step. Multiple sorting and scanning steps compute the cube result. Unlike the case for distributive cubes, fine granularity results are not used to compute coarse granularity results; the aggregates are computed separately at each granularity.

The CPU performance measurements given below were generated on an UltraSparc 2 (200MHZ) running Solaris. The machine used had 128MB of RAM, and in all cases, the input relations used were able to fit in RAM. We would expect analogous results for larger tables using the divide and conquer approach of [RS97]. The reported time is the CPU time reported by the operating system, which was always very close to the elapsed time. CPU time measurement commenced after the input was read, and the writing of the output result was suppressed in order to avoid introducing I/O cost.

We ran four multi-feature cube queries, numbered B1 through B4. For comparison purposes, we also ran two datacube queries, numbered C1 and C2, with distributive aggregate functions. B1 and B2 are distributive, while B3 and B4 are holistic. All queries were run over an input with seven attributes, including four `CUBE BY` attributes (G1–G4) and three aggregated attributes (A1–A3). We used three different data sources for the input: (a) a uniform, randomly generated data set, (b) data generated according to the TPC-D benchmark [Tra95], and (c) data from real-world measurements of cloud coverage over the globe for a period of one month [HWL94]. The meaning of each attribute is given below.

Table	G1	G2	G3	G4	A1	A2	A3
TPC-D	Part	Supplier	Date	Return code	Discount	Quantity	Price
Cloud	Latitude	Longitude	Day	Hour	Weather code	low-cloud amount	total cloud amount
Random	Uniformly generated						

Query B1 outputs the maximum A1 value, as well as the maximum A3 value among all tuples having maximum A1 value in the group. Query B2 outputs the maximum A1 value, and then among tuples having maximum A1 value in the group it selects two subgroups: those with minimum A2 value, and those with maximum A2 value. (The minimum and maximum A2 values are also output.) For each of the three groups the maximum A3 value within the group is output. Query B3 finds (and outputs) the maximum A1 value within the group, and then computes (and outputs) the maximum A3 value in three subgroups: those tuples that have an A1 value more than 25% of maximum, those tuples that have an A1 value more than 50% of maximum, and those tuples that have an A1 value more than 75% of maximum. Query B4 finds the maximum A1 value and the maximum A2 value within a group. It then computes the maximum A3 value for all tuples in the group that have both an A1 value greater than 50% of maximum, and an A2 value greater than 50% of maximum. The maximum A1 and A2 values, together with the computed maximum A3 value are output; if no input tuples satisfy the conditions, then no output tuple is generated. Query C1 outputs the maximum A1 and A3 values, and Query C2 outputs the maximum A1, A2, and A3 values, along with the minimum A1, A2 and A3 values.

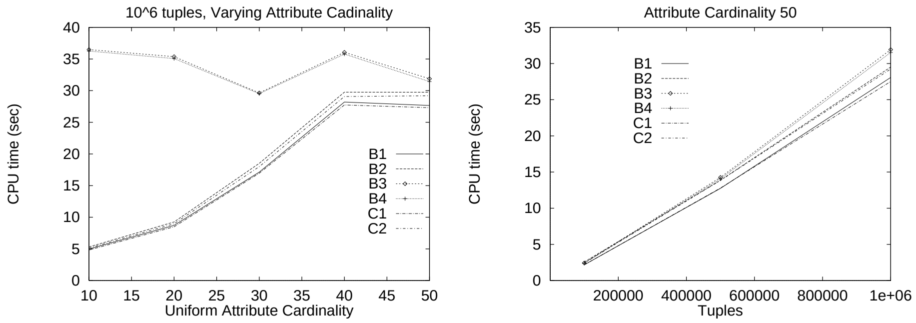

Fig. 3. Performance for Random Datasets

We present four performance graphs. The first two, given in Figure 3 show the performance on the random datasets.

In the left graph, we fix the input size at 10^6 tuples, and vary the (uniform) attribute cardinality. For low cardinality, the cube is *dense*, since the number of combinations of input `CUBE BY` attributes is 10^4, much smaller than the number of tuples. For high cardinality, the cube is sparse ($50^4 > 10^6$). The graph shows that for sparse data, at the right edge of the graph, the aggregate functions perform comparably, within a fairly narrow range. However, for dense data, the distributive functions perform substantially better than the holistic functions. The reason for this behavior is the initial combining step that is performed for distributive aggregates but not for holistic aggregates. In the distributive case, the initial data set will collapse to 10^4 tuples after combining when the cardinality is 10, meaning that all of the subsequent work is performed on 10^4 tuples rather than 10^6 tuples.

In the right graph, we show how the performance scales with the number of tuples when the attribute cardinality is 50. As the number of tuples increases, the density gradually increases, and so the holistic cubes begin to be more expensive at the right edge of the graph. If one looks closely, the other curves are divided into two groups with the curves in each group being very close to one another. One group consists of B1 and C1, and the other consists of B2 and C2. Both B1 and C1 have two output aggregates, while both B2 and C2 have six output aggregates. Thus, the complexity seems to be determined more by the *number* of output columns than by whether the distributive aggregate function is simple or complex. In other words, *there is negligible apparent overhead for evaluating a complex distributive multi-feature cube when compared with a simple datacube having the same number of output columns.*

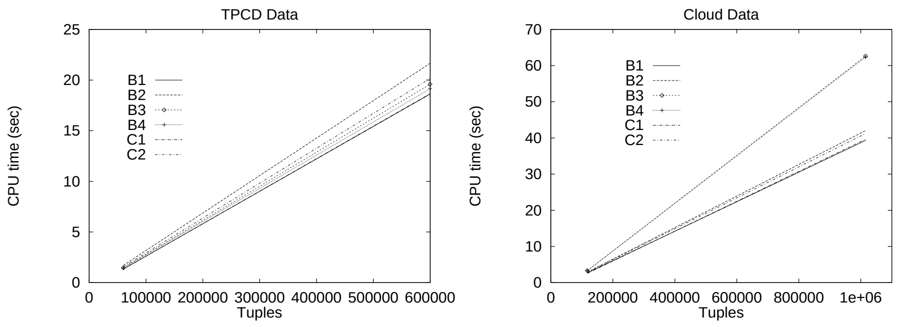

Fig. 4. Performance for TPC-D and Cloud Datasets

The remaining two graphs, given in Figure 4, show the performance on the TPC-D and Cloud data sets. The TPC-D data has similar performance characteristics to the random data set. The Cloud data set also shows similar behavior, with density becoming an issue at the right edge of the graph; at this point only 70% of the tuples remain after the initial combining step.

While holistic multi-feature cubes are sometimes significantly more expensive than distributive cubes, we can see that computing them involves a CPU cost that is relatively manageable.

8 Conclusions

We have considered multi-feature cube queries, a natural class of complex queries involving multiple dependent aggregates computed at multiple granularities, that are practically very useful. We have classified multi-feature cube queries according to the degree of incrementality by which coarser granularity results can be computed from finer granularity results. We have identified syntactic sufficient conditions that allow us to recognize multi-feature cubes that are distributive or algebraic. We have shown that a number of existing algorithms can be used for the evaluation and optimization of distributive (or algebraic) multi-feature cubes without any increase in I/O cost, and with a negligible increase in CPU cost. This is an important subclass of multi-feature cube queries that is no more difficult to compute than datacube queries, and which includes many sophisticated queries not easily expressed as datacube queries.

Other authors have introduced new algebraic operators and/or syntaxes for multidimensional data analysis [AGS97, LW96]. However, none of these proposals considers the issue of multiple dependent aggregates within a group. As a result, the specification of most of the examples presented in this paper using their approaches would require multiple views or subqueries, leading to the problems outlined in [CR96].

Acknowledgements

We would like to thank Ted Johnson for his comments on this paper. The research of Kenneth A. Ross and Damianos Chatziantoniou was supported by a grant from the AT&T Foundation, by a David and Lucile Packard Foundation Fellowship in Science and Engineering, by a Sloan Foundation Fellowship, by an NSF Young Investigator Award, and by NSF CISE award CDA-9625374.

References

[AAD+96] S. Agarwal, R. Agrawal, P. M. Deshpande, A. Gupta, J. F. Naughton, R. Ramakrishnan, and S. Sarawagi. On the computation of multidimensional aggregates. In *Proceedings of VLDB*, pages 506–521, 1996.

[AGS97] R. Agrawal, A. Gupta, and S. Sarawagi. Modeling multidimensional databases. In *Proceedings of IEEE ICDE*, 1997.

[CR96] D. Chatziantoniou and K. A. Ross. Querying multiple features of groups in relational databases. In *Proceedings of VLDB*, pages 295–306, 1996.

[GBLP96] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Datacube : A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. In *Proceedings of IEEE ICDE*, pages 152–159, 1996. Also available as Microsoft Technical Report MSR-TR-95-22.

[HWL94] C. J. Hahn, S. G. Warren, and J. London. Edited synoptic cloud reports from ships and land stations over the globe, 1982-1991. Available from http://cdiac.esd.ornl.gov/cdiac/ndps/ndp026b.html, 1994.

[LW96] C. Li and X. S. Wang. A data model for supporting on-line analytical processing. In *Proceedings of CIKM*, pages 81–88, 1996.

[RS97] K. A. Ross and D. Srivastava. Fast computation of sparse datacubes. In *Proceedings of VLDB*, pages 116–125, 1997.

[RSC97] K. A. Ross, D. Srivastava and D. Chatziantoniou. Complex aggregation at multiple granularities. AT&T Technical Report, 1997.

[Tra95] Transaction Processing Performance Council (TPC), 777 N. First Street, Suite 600, San Jose, CA 95112, USA. *TPC Benchmark D (Decision Support)*, May 1995.

[ZDN97] Y. Zhao, P. M. Deshpande, and J. F. Naughton. An array-based algorithm for simultaneous multidimensional aggregates. In *Proceedings of ACM SIGMOD*, pages 159–170, 1997.