Multidimensional Benchmarking in Data Warehouses

Akiko Campbell[†‡] (akiko.campbell@gmail.com),
Xiangbo Mao[‡¶*] (xiangbom@sfu.ca),
Jian Pei[‡§] (jpei@cs.sfu.ca),
Abdullah Al-Barakati[§] (aaalbarakati@kau.edu.sa)
[†]LifeLabs Medical Laboratories, Burnaby, BC, Canada
[‡]Simon Fraser University, Burnaby, BC, Canada
[¶]Zhejiang University, Hangzhou, Zhejiang, China
[§]King Abdulaziz University, Jeddah, Saudi Arabia

Abstract. Benchmarking is among the most widely adopted practices in business today. However, to the best of our knowledge, conducting multidimensional benchmarking in data warehouses has not been explored from a technical efficiency perspective. In this paper, we formulate benchmark queries in the context of data warehousing and business intelligence, and develop algorithms to answer benchmark queries efficiently. Our methods employ a few interesting ideas and the state-of-the-art data cube computation techniques to reduce the number of aggregate cells that need to be computed and indexed. An empirical study using the TPC-H and the Weather data sets demonstrates the efficiency and the scalability of our methods.

Keywords: Benchmarking, data cubes, data warehouse, business intelligence

1. Introduction

Organizations conduct benchmarking for continuous improvement. As a first step, they want to identify if there are areas where they are not performing as well as others. For example, an analyst is interested in finding the performance of the senior sales representatives in Asia by measuring the average sales amount

[*]Corresponding author. Address: 8888 University Drive, Burnaby, BC, Canada V5K1Y7; Tel:+1-604-442-6805

per representative. The analyst may then be interested in finding factors that make up the context in which the performance of the senior sales representatives is measured. These factors may include the product lines, the purchasing customer industry, and the transaction time in a month. The analyst wants to find other sales groups that are performing significantly better than the group of senior sales representatives in Asia in some contexts. For example, some answers interesting to the analyst may be, "compared to the group in question, the sales representatives in North America outperforms the group most for selling laptops to financial business customers during the first 2 quarters of the year". This kind of questions are also known as *benchmarking analysis* in business analytics, since they try to find a benchmark for a query group.

The recognition of benchmarking as a useful management tool was formalized in early 1980s when Xerox employed benchmarking as part of its "Leadership through Quality", a program to find ways to reduce manufacturing costs. In 1982, Xerox determined that the average manufacturing cost of copies in Japanese companies was 40-50% of that of Xerox's, and they were able to undercut Xerox's prices effortlessly. As part of the "Leadership through Quality", Xerox established the benchmarking program, which played a major role in pulling Xerox out of trouble in the years to come. Xerox since then has become one of the best examples of the successful implementation of benchmarking [20].

Benchmark queries can be very sophisticated. For example, one may add various constraints to refine search space. Instead of comparing a query group with any group, one may only be interested in those groups that are super-groups, sub-groups or sibling groups of the query group. For instance, the super-groups of senior sales representatives in Asia are the sales representatives in Asia, the senior sales representatives in the world, and all sales representatives in the world. Likewise, the sibling groups of senior sales representatives in Asia are the groups of senior sales representatives in other regions, such as North America, South America, Europe, and the group of junior sales representatives in Asia.

Benchmarking in business is related to egocentric analysis. In egocentric analysis, given a query group, it tries to identify aspects in which the query group is better than its peers. For example, given a group of senior sales representatives in Asia as the query group, the egocentric analysis tries to identify the factors by which this group performs the best compared to the other groups. An answer may look like "compared to the senior sales representatives in other regions, the query group has the best performance in selling desktop computers to education customers". If a query group cannot find a benchmark in a subspace, the subspace is the answer to the egocentric analysis for the query group.

Data warehouses are the essential information infrastructure in modern enterprises. However, to the best of our knowledge, benchmarking effectively and efficiently in data warehouses remains largely unexplored from a technical perspective in data management and analytics. Benchmark queries cannot be answered online using the existing data cube and data warehouse techniques. Even when we compute a whole data cube using all the attributes, in the context of benchmarking, we need to define a query group and then need to search the cube for the answers to the query. It is well recognized that the size of a data cube is exponential with respect to the number of tuples and the dimensionality of the base table.

In this paper, we tackle the problem of efficiently answering benchmark queries. We make a few contributions. First, we formulate benchmark queries technically. To the best of our knowledge, this technical problem has not been studied systematically in literature for computational efficiency. Second, we explore algorithmic approaches to benchmark queries. We develop two approaches. First approach is the Sorted Inverted Index Cube (SIIC); we sort aggregate cells in a cube and explore the idea of inverted index for fast access to the search scope of a query. The second approach is the Dominant Answer Materialization (DAM) approach; we exploit a property of aggregate cells to refine the definition of benchmark queries to remove redundancy in answers, thus achieving a more efficiency. Finally, we conduct an extensive experimental study using both synthetic and real data sets to determine the efficiency of our proposed methods.

The rest of the paper is organized as follows. In Section 2, we define benchmark queries and review related work. In Section 3, we develop a Sorted Inverted Index Cube (SIIC) method. In Section 4, we propose a Dominant Answer Materialization (DAM) approach. We report an empirical evaluation in Section 5. Section 6 concludes the paper.

2. Problem Definition and Related Work

In this section, we first outline some preliminaries in data cube and multidimensional analysis. We then define the benchmark queries formally.

2.1. Preliminaries

We largely follow the notations in the conventional data cube and data warehouse literature [8]. Consider a relational table $T = (TID, A_1, \ldots, A_n, M)$, called *base table*, and an aggregate function f, where TID is a tuple-id attribute to ensure every tuple in the table is unique, $A_1, \ldots, A_n$ are the *dimensions attributes* and M is a *measure attribute*. We assume all dimension attributes are categorical, and the measure attribute can be categorical or numeric. For a tuple $t \in T$, denote by $t.TID$, $t.A_i$ and $t.M$ the values of t for the TID attribute, the dimension A_i, and the measure value of M, respectively.

Let $D = \{A_{i_1}, \ldots, A_{i_l}\}$ be a subset of dimensions, where $1 \leq i_1 < i_2 < \cdots < i_l \leq n$. D is often called a *subspace*. The *cuboid* on D is the group-bys using attributes in D, denoted by C_D. Apparently, C_D is a set of tuples. Note that D may be empty, that is, $D = \emptyset$.

An *aggregate cell* in the cuboid on D is a tuple $c = (*, a_{i_1}, *, a_{i_2}, \cdots, *, a_{i_l}, *, aggr) \in C_D$, where a_{i_j} belongs to the domain of attribute $A_{i_j} (1 \leq j \leq l)$, meta-symbol $*$ denotes that the dimension is generalized, and $aggr = f(\{t.M \mid t.A_{i_j} = a_{i_j}, 1 \leq j \leq l\})$ is the aggregate of all tuples in the group $(*, a_{i_1}, *, a_{i_2}, \cdots, *, a_{i_l}, *)$. To keep our presentation simple, we overload the symbol $c.M = aggr$. Further, we ignore the aggregate cells c if the aggregate group $\{t \mid t.A_{i_j} = a_{i_j}, 1 \leq j \leq l\}$ is empty; that is, it does not contain any tuple in the base table.

We can define a partial order $\prec$ on cuboids. The cuboids $C_{D_1} \prec C_{D_2}$ if $D_1 \subset D_2$. The set of cuboids form a lattice with respect to partial order $\prec$. We can also define a partial order $\prec$ on aggregate cells. The cells $t_1 \prec t_2$ if for each dimension A_i $(1 \leq i \leq n)$, when $t_1.A_i \neq *$, then $t_1.A_i = t_2.A_i$. We also say t_1 is an *ancestor* of t_2 and t_2 a *descendant* of t_1. The set of aggregate cells too forms a lattice with respect to partial order $\prec$. For two aggregate cells t_1 and t_2, t_1 is a *sibling* of t_2 if t_1 and t_2 have identical values in all dimensions except for one in which neither has value "$*$".

A *data cube* is the set of cuboids on all possible subspaces, that is, all subsets of dimensions, including the empty set. Equivalently, a data cube is also the set of all aggregate cells. We denote a data cube by $cube(T, DIM, M, f)$, where T is the base table, DIM is a subset of attributes of T that are used as the dimensions in the cube, M is the measure attribute, and f is the aggregate function.

For two aggregate cells u and v, if there does not exist a dimension A_i such that neither $u.A_i$ nor $v.A_i$ has value "$*$" and $u.A_i \neq v.A_i$, then the *concatenation* of u and v, denoted by $w = u \otimes v$, is an aggregate cell such that for attribute A_i, $w.A_i = u.A_i$ if $u.A_i \neq *$, otherwise $w.A_i = v.A_i$.

Example 1 (Preliminaries)**.** Consider a relation table $T = \{TID$, position, gender, location, salary$\}$ for sales representatives of a company, where position, gender, and location are the dimensions and salary is the measure. Suppose we use $avg()$ as the aggregate function, $c = (*, \text{male}, *, avg())$ is an aggregate cell, which represents the average salary of all male sales representatives of the company.

Consider aggregate cells $u = (\text{senior}, *, *)$, $t = (\text{senior}, \text{male}, *)$ and $t' = (\text{senior}, \text{female}, *)$. We have $u \prec t$, which means u is an ancestor of t, and t is a descendant of u. Further, t and t' are siblings. Aggregate cell $v = (*, \text{male}, \text{North America})$ represents the male sales representatives in North America. We can use the concatenation operator to get all senior male sales representatives in North America; that is, $w = u \otimes v = (\text{senior}, \text{male}, \text{North America})$. ■

2.2. Benchmark Queries

We consider a relational *base table* $T = (TID, A_1, \ldots, A_n, M)$. The attributes of T that will be used in a benchmark query can be divided into three groups: the *unit-id attributes* UID, the *dimension attributes* DIM, and the *measure attributes* M, where $UID \cup DIM \subseteq \{A_1, \ldots, A_n\}$. We do not require that UID and DIM are exclusive, that is, $UID \cap DIM = \emptyset$ is not assumed.

The unit-id attributes are used to group tuples in T into aggregate units. Since the term "group" can mean different things, to avoid confusion, for the rest of this paper, we call a group a *unit.* We consider the data cube formed using the unit-id attributes UID where each aggregate cell in the data cube corresponds to a *unit.* We are interested in comparing the performance of the units.

The dimension attributes are used to conduct multidimensional comparative analysis between two units. The measure attribute is used to calculate aggregates and derive quantity analysis. For the sake of simplicity, we have only one measure attribute in our discussion. However, our methods can be extended to the scenarios where multiple measure attributes are used to derive sophisticated aggregates. We also assume that the measure attribute takes non-negative values. This assumption often is true in many business intelligence applications. For example, measures such as count, sales volume, and amount are often used in practice. Even when a measure can take negative values, we can always normalize the attribute such that the normalized measure attribute has non-negative values.

For each *non-empty unit* that consists of at least one tuple in the base table, using the dimension attributes DIM and the measure attributes M, we can form a data cube, which quantifies the performance of the unit in multidimensional aspects.

Example 2 (Attributes)**.** Consider base table $T = \{$age-group, gender, location, position, education, salary$\}$ that contains properties of employees of a company. For simplicity, we omit the tuple-id attribute.

Suppose we use attributes $UID = \{$age-group, gender$\}$ as the unit-id attributes. That is, we are interested in comparing various groups formed by the group-by operation by these two attributes. For example, (young, male) and (mid-age, *) are two aggregate units.

We use attributes $DIM = \{$location, position, education$\}$ as the dimension attributes. That is, we compare units by those three dimensions. Finally, we use $M = \{$salary$\}$ as the measure attribute. Using the aggregate function average, we can compare the average salary between different units with respect to different office locations, positions, education levels and their combinations. For example, we may find that, for the position "technical support" at location "Vancouver", the age group "mid-age" has a much higher average salary than the age group "young". A possible explanation may be seniority and years of experience. ■

As noted earlier, unit-id attributes and dimension attributes may not be exclusive. That is, an attribute may be both a unit-id attribute and a dimension attribute. Technically, we can always create a copy of an attribute that is used as both a unit-id attribute and a dimension attribute. As such, one copy is used as a unit-id attribute and the other copy is used as a dimension attribute. Therefore, without loss of generality, for the rest of the paper, we assume that the unit-id attributes and the dimension attributes are exclusive.

To compare two aggregate cells c and c', we are interested in the ratio of their measures. In the rest of the paper, we focus on the ratio $\frac{c.M}{c'.M}$. The larger the ration, the better c is. Our discussion can be extended to comparing two aggregate cells in some other ways.

For a unit u, an aggregate cell c defined using the dimension attributes is called an *aspect* of u if $u \otimes c$ is in the cube $Cube(B, UID \cup DIM, M, f)$. Given two units u and v defined using the unit-id attributes and an aggregate cell c defined on the dimension attributes such that c is an aspect of both u and v, $\frac{(u \otimes c).M}{(v \otimes c).M}$ measures the advantage of u over v in aspect c. The larger the ratio, the better u is in c than v. We denote by $R(u/v \mid c) = \frac{(u \otimes c).M}{(v \otimes c).M}$, the *advantage* of u over v in c.

We define a benchmark querie *benchmark query* Q as follows:

– a base table T and the specification of the unit-id attributes UID, dimensions DIM, and measure M;
– a query unit q that is an aggregate cell in the data cube formed by the unit-id attributes UID;
– the search scope, namely ancestors, descendants and siblings; and
– a parameter k.

Let u be a unit on the unit-id attributes and c be an aspect of the query unit q. (u, c) is a top-k answer to the benchmark query Q if:

age-group	gender	location	position	education	Sales
young	M	Vancouver	staff	University	200
young	F	Seattle	manager	College	230
young	F	Seattle	manager	University	220
mid-age	M	Vancouver	staff	College	220
mid-age	M	Seattle	staff	University	200
mid-age	M	Vancouver	manager	University	224

Table 1. Base table of employees of a company.

- u is in the search scope; that is, an ancestor, a descendant, or a sibling of $q.UID$ as specified in the query input;
- $\frac{(u \otimes c).M}{(q \otimes c).M} > 1$; and
- there are at most $k-1$ pairs (u', c') such that u' is also in the search scope, $c' \neq c$ is another aspect of u, and $\frac{(u' \otimes c').M}{(q \otimes c').M} > \frac{(u \otimes c).M}{(q \otimes c).M}$.

The requirement $\frac{(u \otimes c).M}{(q \otimes c).M} > 1$ ensures that u has a non-trivial advantage over q in c and thus is a significant benchmark for q. We ignore the aggregate cells c such that $q \otimes c$ is empty. For each answer (u, c) in the top-k answers, u is called a *benchmark unit*, and the subspace c is called the *benchmark aspect* of u.

Given a benchmark query Q, we want to compute all top-k answers to the query. When there are multiple answers (u, c) with the same advantage over the query unit q, we will return more than k answers.

Example 3 (Benchmark query)**.** Consider base table $T = \{$age-group, gender, location, position, education, salary$\}$ that contains properties of a company. Table 1 shows the base table. We use $avg()$ as the aggregate function. Let $UID = \{\text{age-group}, \text{gender}\}$, $DIM = \{\text{location}, \text{position}, \text{education}\}$, and $M = \{\text{sales}\}$.

Suppose the query unit is $q = $ (young, Male), $k = 2$, and the search scope is siblings. The top-2 answers are ((young, Female), $(*, *, *)$) (ratio: $\frac{225}{200} = 1.125$), ((mid-age, Male), (Vancouver, *, University)) (ratio: $\frac{224}{200} = 1.12$). $(*, *, *)$ and (Vancouver, *, University) are the corresponding aspects. ■

Aggregate functions can be categorized into two types: monotonic aggregates and non-monotonic aggregates. An aggregate function f is *monotonic* if for any aggregate cells c_1 and c_2 such that $c_1 \prec c_2$, $f(c_1) \geq f(c_2)$. An aggregate function is *non-monotonic* if it does not have this property. For example, if the measure attribute only has non-negative values, then aggregate functions $sum()$ and $count()$ are monotonic. Aggregate function $avg()$ is non-monotonic.

For a monotonic aggregate function, answering a benchmark query is straightforward, since the apex cell $(*, *, \ldots, *)$ always has the largest aggregate value and thus the largest ratio. However, using a monotonic aggregate function in a benchmark query is uninteresting because it does not lead to any notable knowledge discovery. In this paper, we assume that the aggregate functions used are non-monotonic, such as, $avg()$, and aggregate values are positive. The methods developed here can be applied to cases where aggregate functions are monotonic.

2.3. Related Work

Benchmark queries are related to iceberg queries, gradient analysis, and discovery-driven OLAP. We briefly review the existing studies and how benchmark queries differ from them.

In online analytics processing (OLAP) in data warehouses, iceberg queries [7] compute the aggregate cells whose aggregate values are over a user defined threshold. For example, in a data cube of sales data, an iceberg query may return all the aggregate cells whose total sales amount is over 100 thousand dollars.

Iceberg queries have been extensively studied. A focus is on efficient algorithms for answering iceberg queries. For example, Beyer and Ramakrishan [2] proposed algorithm BUC which computes iceberg cubes with monotonic aggregate functions. Han *et al.* [9] developed a method for computing iceberg queries with non-monotonic aggregate functions. Ng *et al.* [16] investigated iceberg queries on distributed systems. Chen *et al.* [5] explored iceberg cube computation in shared-nothing clusters. Lo *et al.* [15] extended iceberg queries to sequence data. Recently, He *et al.* [10] used patterns as "dimensions" in iceberg queries on sequences. Chen *et al.* [4] extended iceberg queries to graphs.

Although both iceberg queries and benchmark queries are concerned with aggregates, they are fundamentally different. Iceberg queries do not use any query unit and do not compare aggregate cells. Further, while Iceberg queries are often used as the first step for materializing a scope for further analysis, benchmark queries are tools for more focused analysis on a target unit. Benchmark queries cannot be answered by a straight application of iceberg query methods.

In gradient analysis [6, 13], given a probe aggregate cell q, we want to find all pairs of aggregate cells (q, v) such that q is an ancestor of v and the change of aggregates from q to v is significant, guarded by a gradient threshold. For example, given that the average house price in Vancouver is 1.1 million dollars as the probe cell, using gradient analysis we can find all the sub-regions of Vancouver where the average house price is 20% higher than 1.1 million dollars. Gradient analysis has been found useful in business intelligence [3, 17]. More efficient and effective algorithms were proposed [19]. We note that gradient analysis can also be extended to search for pairs (v, q), where q is a descendant of v.

There are some similarities between gradient analysis and benchmark queries. First, both gradient analysis and benchmark queries use a query aggregate cell, and find interesting aggregate cells when compared to the query cell. Second, both gradient analysis and benchmark queries use the aggregate ratios as the significance measure. Third, both gradient analysis and benchmark queries can search ancestors and descendants of the query cell. However, the two types of queries have some fundamental differences. Gradient analysis does not separate the unit attributes and the dimension attributes. Gradient analysis in this sense can be regarded as a special case of benchmark queries where the set of dimension attributes DIM is empty. Also, the business objectives for the two are also very different. Benchmark queries facilitate more detailed multidimensional analysis for comparing the query unit with the other units in the intended search scope.

Sarawagi *et al.* [18] developed the notion of discovery-driven exploration of OLAP cube. The main idea is to identify anomalies within a data cube and provide proper indicators in the corresponding aggregate cells. Both discovery-driven exploration of OLAP cube and benchmark queries want to find significant exceptions. However, discovery-driven exploration does not focus on one query cell. It instead considers all descendant cells for each aggregate cell. Further, the objectives in the two problems are very different. Discovery-driven exploration aims to provide navigation guidance for users to browse interesting regions of a cube, while benchmark queries compares a query unit with the other units. Thus, the measures used are very different.

Common forms of benchmarking methods lend from economic efficiency analysis which involve parametric and non-parametric techniques. The primary objective of both is to measure the technical efficiency which is defined as the ability of a producer to produce maximum output from a given set of inputs. Technical efficiency thus is translated as the success indicator of performance measure by which producers are evaluated. Given the importance of technical efficiency analysis, several models of frontiers have been developed. The frontier models are based on the premise that efficient producers operate on the production frontier using the most efficient technology available, while inefficient producers operate below the production frontier and the level of inefficiency is measured by the deviation from the frontier [12]. The major advantage of this method is that it allows the test of hypothesis concerning the goodness of fit of the model. The stochastic aspect of the model allows it to handle measurement problems appropriately and other stochastic influences that would otherwise show up as causes of inefficiency [11]. However, the major drawback is that it requires specification of technology, which may be restrictive in most cases [12].

Data Envelopment Analysis (DEA) is a non-parametric linear programming technique widely used in the operations research and management science literature [1]. DEA estimates the cost level that an efficient organization should be able to achieve in a particular market. The model seeks to determine an envelopment surface, also referred to as the efficiency frontier. Rather than estimating the impact of different cost drivers, DEA establishes an efficiency frontier (taking account of all relevant variables) based on the "envelope" of observations. Each organization is then assigned an efficiency score based on its proximity to the estimated efficiency frontier. With DEA, the efficient frontier (determined by the efficient organizations in the sample) is the benchmark against that the relative performance of organizations is measured.

As pointed out earlier, conducting benchmarking effectively and efficiently using scalable computational technology, particularly on readily available data warehouse infrastructure, has not been explored in literature. Bridging the gap between business needs and the technology motivates this study.

3. A Sorted Inverted Index Cube (SIIC) Method

With the advanced data warehousing techniques, we can materialize a multidiemsional data cube. We assume a data cube materialization method $Cube(B, \{A_1, \ldots, A_n\}, M, f)$ that computes a data cube from a multidimensional table B using the attributes $A_1, \ldots, A_n$ as dimensions, M as the measure, and aggregate function f. In our experiments, we use the BUC algorithm [2] to materialize a data cube.

For each possible unit u, let B_u be the set of tuples in the base table that belong to u. That is, $B_u = \{t \mid t \in B \wedge u \preceq t\}$. Given a query unit q, a benchmark query compares the data cube $Cube(B_q, DIM, M, f)$ and $Cube(B_u, DIM, M, f)$ for every unit u in the search scope.

To facilitate answering benchmark queries, we can materialize $Cube(B_u, DIM, M, f)$ for every unit u. This is equivalent to materializing the whole data cube $Cube(B, UID \cup DIM, M, f)$, since we also need to find all units using attributes UID. The remaining problem is how to find answers in the whole data cube.

A naïve method would be to, given unit q, we search every unit u in the scope and compute the advantage of u over q on every possible aggregate cell c in the set of attributes DIM. However, the search scope of a query unit may contain an exponential number of units. Also, there are another exponential number of aggregate cells in the set of attributes DIM. As a result, checking the advantage for every unit u in every subspace c is very time consuming. To address this, we want to organize units and aspects systematically such that the search can ignore many non-promising aggregate cells. To that end, we propose a method based on simple ideas.

3.1. Inverted indices for Fast Search

We use two simple ideas to facilitate fast search.

The first idea is to sort all aggregate cells in the cube $Cube(B, UID \cup DIM, M, f)$ in the aggregate value descending order. We search aggregate cells in this order for query answering. Using this order, we visit the aggregate cells of larger aggregate values earlier and thus heuristically we have a better chance of getting aggregate cells that have more significant advantage over the query cell.

Let $\prec_{aggr}$ be the aggregate value descending order on all aggregate cells in $Cube(B, UID \cup DIM, M, f)$. If there are two or more aggregate cells having the same aggregate value, the tie can be broken arbitrarily. For any aggregate cells u and v, if $u \prec_{aggr} v$, then $u.M \geq v.M$.

The second idea is to facilitate the visit of aggregate cells in the search scope using inverted indices. For every unit-id attribute, we maintain an inverted index for each value of the domain to record the list of aggregate cells containing the value. Suppose a_{ij} is a value in the domain of unit-id attribute A_i. We build an inverted index $Index_{a_{ij}}$ which is a list of aggregate cells $u \in Cube(B, UIC \cup DIM, M, f)$ such that $u.A_i = a_{ij}$. All the aggregate cells in every inverted index list are sorted according to the order $\prec_{aggr}$.

We can retrieve all aggregate cells of cube $Cube(B_q, DIM, M, f)$ using the inverted indices efficiently in a way similar to merge-sort. Let q be the query unit, and $q.A_{i_1}, \ldots, q.A_{i_l}$ are the unit-id attribute values that are not $*$. To find all aggregate cells for the unit q, we only need to search the inverted indices $Index_{q.A_{i_1}}, \ldots, Index_{q.A_{i_l}}$ and find all aggregate cells c such that c appears in every list $Index_{q.A_{i_j}}$ and has the value $*$ in all other unit-id attributes. Since we scan the inverted index lists in the order of $\prec_{aggr}$, we can find all aggregate cells in unit q in one scan.

We also show that retrieving all unit aggregate cells in the search scope, that is, ancestors, descendants and siblings, can also be conducted efficiently using the inverted index in a way similar to merge-sort. Let q be the query unit, and $q.A_{i_1}, \ldots, q.A_{i_l}$ be the unit-id attribute values that are not $*$. To search for the ancestor units and their aggregate cells, we scan the inverted indices $Index_{q.A_{i_1}}, \ldots, Index_{q.A_{i_l}}$ in a synchronized manner. Except for unit $(*, \ldots, *)$, which can be checked separately as a special case, an aggregate cell c is an ancestor of q if (1) c appears in at least one of the inverted index $Index_{q.A_{i_1}}, \ldots, Index_{q.A_{i_l}}$; and (2) $c.A_{i_j} = *$ if c does not appear in $Index_{q.A_{i_j}}$.

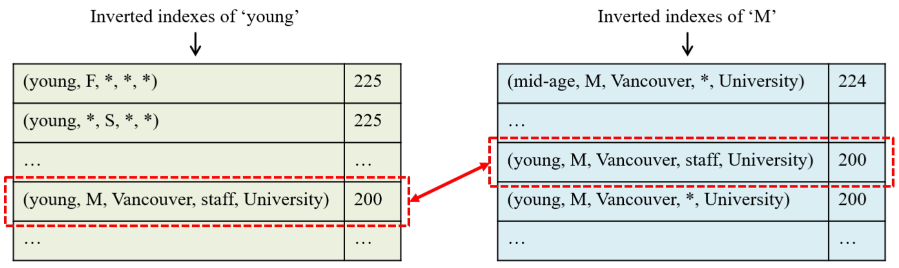

Inverted indexes of 'young'
(young, F, *, *, *) 225
(young, *, S, *, *) 225
… …
(young, M, Vancouver, staff, University) 200
… …
Inverted indexes of 'M'
(mid-age, M, Vancouver, *, University) 224
…
(young, M, Vancouver, staff, University) 200
(young, M, Vancouver, *, University) 200
… …

Fig. 1. Example of SIIC, using inverted indices for values "young" and "M".

Again, since we scan the inverted indices in the order of $\prec_{aggr}$, we can find all ancestor units of q and their aggregate cells in one scan.

To find all descendant units, we search the inverted indices $Index_{q.A_{i_1}}, \ldots, Index_{q.A_{i_l}}$ and find all cells c such that c appears in every inverted index $Index_{q.A_{i_j}}$ and takes a non-* value in at least one unit-id attribute other than $A_{i_1}, \ldots, A_{i_l}$. To find all siblings of q, we search the inverted indices $Index_{q.A_{i_1}}, \ldots, Index_{q.A_{i_l}}$ and find all cells c such that (1) c appears in every inverted index $Index_{q.A_{i_j}}$ except for one, say $Index_{q.A_{i_{j_0}}}$; (2) $c.A_{i_{j_0}} \neq q.A_{i_{j_0}}$, and $c.A_{i_{j_0}} \neq *$; and (3) $c.A_{i_j} = *$ if $q.A_{i_j} = *$. Clearly, the above two searches can be achieved in one scan of the inverted indices.

Example 4 (SIIC)**.** We use Table 1 as the example. Let $avg()$ be the aggregate function. Let $UID = \{\text{age-group}, \text{gender}\}$, $DIM = \{\text{location}, \text{position}, \text{education}\}$, and $M = \{\text{sales}\}$.

First, we build an inverted index for each value in the domain of every unit-id attribute. Two examples of inverted indices are shown in Figure 1.

Suppose the query unit is $q = $ (young, M), we can find that (young, M, Vancouver, staff, University) appears in both the inverted indices of young and M, thus (young, M, Vancouver, staff, University) must be an aggregate cell of unit q. Similarly, we can easily find the set of all aggregate cells of unit q, $\{$(young, M, Vancouver, staff, University) : 200, (young, M, Vancouver, *, University) : 200, $\ldots\}$, using the inverted indices.

To find all aggregate cells of the ancestors, descendants and siblings of q, we can apply the similar technique to the search scope of q. For example, to find all the aggregate cells of the sibling unit (young, F), we only need to check the aggregate cells appearing in both the inverted indices for "young" and "F". We have all aggregate cells sorted in the search scope $\{$(young, F, *, *, *) : 225, (mid-age,M), (Vancouver,*,University) : 224, $\ldots\}$.

We will continue and complete the query answering process in Example 5. ■

3.2. Pruning

Since we scan the aggregate cells in the aggregate value descending order, we maintain the top-k answers we have seen so far. We have the following property.

Lemma 1. *Consider an aggregate cell c on the dimension attributes such that*

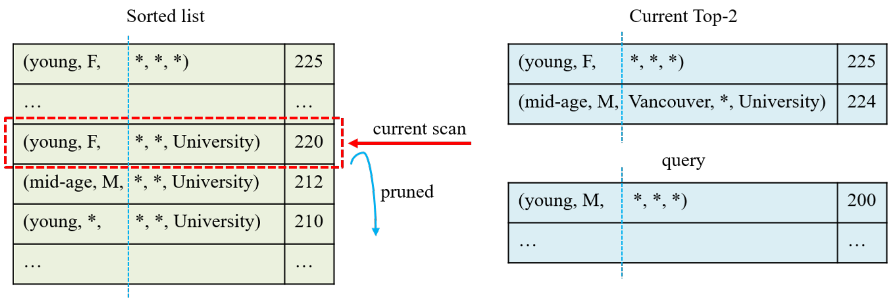

Sorted list
(young, F, | *, *, *) | 225
… | …
(young, F, | *, *, University) | 220
(mid-age, M, | *, *, University) | 212
(young, *, | *, *, University) | 210
… | …
current scan
pruned
Current Top-2
(young, F, | *, *, *) | 225
(mid-age, M, | Vancouver, *, University) | 224
query
(young, M, | *, *, *) | 200
… | …

Fig. 2. Example of SIIC with pruning.

$q \otimes c$ is not empty. For two units u and u' such that $u \otimes c \prec_{aggr} u' \otimes c$, $\frac{(u \otimes c).M}{(q \otimes c).M} \geq \frac{(u' \otimes c).M}{(q \otimes c).M}$.

Proof. We only need to recall that, if $u \otimes c \prec_{aggr} u' \otimes c$, $(u \otimes c).M \geq (u' \otimes c).M$ and the assumption that the aggregate values are positive. ■

Using the result above, for any aggregate cell c on the dimension attributes such that c is an aspect of q, that is, $q \otimes c$ is not empty, if we meet the condition where an aggregate cell $v = u \otimes c$ such that (u, c) is not qualified as a top-k answer among the aggregate cells processed so far, then any pairs (u', c) to be scanned later is not qualified either, and thus c can be pruned.

Let v be the current aggregate cell we are considering in the inverted indices. For any aspect c of q, if $\frac{v.M}{(q \otimes c).M}$ is less than the top-k answers we have seen so far, then no aggregate cells after v in the sorted list can form a pair (u, c) such that $v = u \otimes c$ and (u, c) is qualified as a top-k answer. In this case, the aspect c can be pruned as well.

Using the above pruning rules, we can prune the aspects of q, that is, the aggregate cells in cube $Cube(B_q, DIM, M, f)$. Once all aspects of q are determined to be either included in the current top-k answers or pruned, the search can terminate and the current top-k answers can be returned as the final answers to the benchmark query.

Example 5 (SIIC cont'd)**.** We incorporate pruning techniques into Example 4.

Suppose the query unit is q = (young, M), and we want to find the top-2 benchmarking answers. Assume that we have {(young, M, *, *, *) : 225, (mid-age, M, Vancouver, *, University) : 224} as the current top-2 benchmarks, as illustrated in Figure 2. We are scanning the aggregate cell {(young, F, *, *, University) : 220}. It is easy to verify that {(young, F, *, *, University) : 220} is not qualified as a top-2 answer. We also know that every unit cell appearing after {(young, F)} has a smaller aggregate value than {(young, F, *, *, University) : 220}. Thus, all the following unit cells compatible with {(*, *, University)} can be pruned. ■

4. A Dominant Answer Materialization (DAM) Method

The SIIC method uses some simple yet efficient techniques to accelerate answering benchmark queries. However, there is one severe drawback; that is, in the worst case, we still have to go through the list of all aggregate cells of the whole data cube $Cube(B, UID \cup DIM, M, f)$. This can incur a significant cost when the data cube is large. In this section, we develop a new method that can answer benchmark queries efficiently if the search scope does not involve siblings.

4.1. Search Scope of Ancestors

We consider the search scope of ancestors first and discuss how to address the search scope of descendants later.

Consider a query unit q and a unit u that is an ancestor of q; that is, $u \prec q$. Then, u is called a *maximal unit* of q with respect to aspect c if c is an aspect of both q and u, and there does not exist another ancestor u' of q such that $\frac{(u' \otimes c).M}{(q \otimes c).M} > \frac{(u \otimes c).M}{(q \otimes c).M}$. We observe the following.

Theorem 1 (Monotonicity)**.** *Given a unit q, if a unit u is a maximal unit of q with respect to aspect c, then for any unit q' such that $u \prec q' \prec q$, u is also a maximal unit of q' with respect to c.*

Proof. We prove by contradiction. Assume that u is not a maximal unit of q' with respect to c. Then, there exists another unit u' such that $u' \prec q'$ and $\frac{(u' \otimes c).M}{(q' \otimes c).M} > \frac{(u \otimes c).M}{(q' \otimes c).M}$.

Since $u' \prec q'$ and $q' \prec q$, we have $u' \prec q$. Since $u' \prec q'$, $\frac{(u' \otimes c).M}{(q' \otimes c).M} > \frac{(u \otimes c).M}{(q' \otimes c).M}$, and the measure values are non-negative, we have $(u' \otimes c).M > (u \otimes c).M$. Consequently, we have $\frac{(u' \otimes c).M}{(q \otimes c).M} > \frac{(u \otimes c).M}{(q \otimes c).M}$. That is, u is not a maximal unit of q with respect to c. A contradiction. ∎

Theorem 1 suggests a useful hint for answering benchmark queries. Multiple query units may share a common aggregate unit as an answer for benchmark queries. To answer benchmark queries efficiently, we can precompute those aggregate units and the associated aspects that may be answers to benchmark queries. With that in mind, the problem now is, for an aggregate unit u, which query units may take u as a possible answer to benchmark queries, and with respect to which aspects? The following lemma answers this question.

Lemma 2. *For aggregate units u and v such that $u \prec v$, let c be an aspect of both u and v. Then, u is not a maximal unit of v with respect to c if:*

1. *there exists an ancestor $u' \prec u$ such that $(u' \otimes c).M > (u \otimes c).M$; or*
2. *there exists a descendant u'' such that $u \prec u'' \prec v$ and $(u \otimes c).M < (u'' \otimes c).M$.*

Proof. If there exists an ancestor $u' \prec u$ such that $(u' \otimes c).M > (u \otimes c).M$, then $R(u'/v \mid c) > R(u/v \mid c)$. If there exists a descendant u'' such that $u \prec u'' \prec v$ and $(u \otimes c).M < (u'' \otimes c).M$, then $R(u''/v \mid c) > R(u/v \mid c)$. In both cases, u is not a maximal unit of v with respect to c. ∎

According to the first statement in Lemma 2, in order to answer benchmark queries whose search scope is ancestors, we do not need to store the whole data cube $Cube(B, UID \cup DIM, M, f)$. Instead, we only need to store those aggregate units u and aspects c whereby there does not exist another unit u' and c' and $(u \otimes c).M < (u' \otimes c).M$. In other words, we only need to store those units and aspects whose measure values are not dominated by any of their ancestors.

For an aggregate unit u and aspect c, we call (u, c) a *dominant answer* if there does not exist another unit u' and c' and $(u \otimes c).M < (u' \otimes c).M$. Thus, to answer any benchmark query, we only need to materialize all dominant answers.

Once all dominant answers are materialized, we can organize those dominant answers using inverted indices as described in the SIIC method. The query answering method remains the same. The second statement in Lemma 2 guarantees the correctness. The major saving in the DAM method is that we do not need to store or search any non-dominant answers.

The last remaining is how to compute the dominant answers. A brute-force method would be to compute a full data cube and then select the dominant answers from all the aggregate cells. Since we are concerned with groups of aggregate cells with different measure values, we can adopt the quotient cube method [14].

The quotient cube method [14], instead of computing all the aggregate cells in a cube, it groups the aggregate cells according to the tuples in the base table that contribute most to the aggregate of the cells. For an aggregate cell u, the method considers the set of descendant tuples in the base table $cov(u) = \{t \mid u \prec t, t \in B\}$. If two aggregate cells u_1 and u_2 share the identical set of descendant tuples in the base table; that is, $cov(u_1) = cov(u_2)$, then the two cells are allocated to the same quotient group. It was shown that each quotient group has a unique upper bound, which is also in the group [14]. In other words, if there are aggregate cells u_1 and u_2 such that $cov(u_1) = cov(u_2)$ but $u_1 \not\prec u_2$ and $u_2 \not\prec u_1$, then there exists another aggregate cell u such that $u \prec u_1$, $u \prec u_2$ and $cov(u) = cov(u_1) = cov(u_2)$.

The quotient group technique thus is suitable for answering benchmark queries. We only need to materialize the upper bounds of the quotient groups that are dominant answers.

Example 6 (DAM)**.** We continue with Table 1. We assume that the query unit is $q =$ (young, M) and use average ($avg()$) as the aggregate measure.

The set of ancestors of the query unit is $\{$(*, M), (young, *), (*, *)$\}$. It is easy to verify that $u =$ (*, M) is a maximal unit of q with respect to the aspect $c =$ (Vancouver, *, University), and $u =$ (*, *) is a maximal unit of q with respect to the aspect $c =$ (Vancouver, staff, *).

According to the base table, ((mid-age, M), (Vancouver, *, University)) is a dominant answer since there does not exist a unit u' that has a greater aggregate value than avg((mid-age, M) $\otimes$ (Vancouver, *, University)).

As an example of quotient cube based on the base table, we can verify that cov(mid-age, M, Vancouver, manager, *) $=$ cov(mid-age, M, *, manager, University); they have the same set of descendant tuples in the base table. Thus, these two aggregate cells are in the same quotient group. Finally, (mid-age, M, *, manager, *) is the upper bound of the group.

With the aid of the quotient cube algorithm, we can materialize all dominant answers from the quotient groups in Table 1; that is, $\{$(young, F, *, *, *), (mid-age, M, Vancouver, *, University), $\ldots\}$. Unlike the SIIC method, we only store the dominant answers with the DAM method; this reduces both the search space as well as time. Once a query is given, we can use the inverted indices to answer the query efficiently. ■

4.2. Search Scope of Descendants

We now consider benchmark queries with the search scope of descendants. Given a query unit q and a unit u that is a descendant of q; that is, $u \succ q$. Then, u is called a *maximal unit* of q with respect to aspect c if c is an aspect of both q and u, and there does not exist another descendant u' of q such that $\frac{(u' \otimes c).M}{(q \otimes c).M} > \frac{(u \otimes c).M}{(q \otimes c).M}$. We have the following result similar to Theorem 1.

Corollary 1 (Monotonicity)**.** *Given a unit q, if a unit u is a maximal unit of q with respect to aspect c, then for any unit q' such that $u \succ q' \succ q$, u is also a maximal unit of q' with respect to c.* ■

We also have a result similar to Lemma 2.

Corollary 2. *For aggregate units u and v such that $u \succ v$, let c be an aspect of both u and v. Then, u is not a maximal unit of v with respect to c if*

1. *there exists a descendant $u' \succ u$ such that $(u' \otimes c).M > (u \otimes c).M$; or*
2. *there exists an ancestor u'' such that $u \succ u'' \succ v$ and $(u \otimes c).M < (u'' \otimes c).M$.*

We thus have a method similar to that for the search scope of ancestors.

5. An Empirical Study

We present an empirical study in this section. The algorithms were implemented with Python 2.7 running with PyPy[1] JIT optimization. PyPy is an advanced just-in-time compiler, which provides about 10 times faster running time and additional scalability for our algorithms than the standard Python. All experiments were conducted using an Intel Core i7-3770 3.40GHz CPU, 16GB memory, and a 1TB HDD running Ubuntu 14.04.

5.1. Data Sets and Experiment Settings

We evaluated our algorithms on both synthetic data and real data.

- **TPC-H benchmark (synthetic data)**. TPC-H (as of the experiemtns, we used TPC-H v2.17.1) is a widely used benchmark that consists of a suite of business oriented ad-hoc queries and concurrent data modifications. TPC-H has 8 separate and individual base tables. We used the joined results of table PART and table LINEITEM as the evaluation base table.
- **Weather data set (real data)**. The weather data set[2] contains $1,015,367$ tuples with attributes including station-id, longitude, latitude, solar-altitude, present-weather, day, hour, weather-change-code, and brightness.

In our experiments, 100 queries were randomly generated for each data set. Each experiment was conducted 10 times, reporting the average values. Using the avg() as the aggregation function, we compare the following methods in the experiments.

[1] http://www.pypy.org/
[2] http://cdiac.ornl.gov/ftp/ndp026b/

Method	Dimensionality of UID	2	3	4	5	6	7	8	9	10
SIIC / SIICP	Computed ($\times 10^6$)	0.4	0.9	2.3	3.5	5.2	6.2	7.5	9.8	12
	Indexed ($\times 10^5$)	0.2	0.4	1.1	2.1	3.6	4.3	6.4	8.0	9.6
DAM	Computed ($\times 10^5$)	0.9	1.7	2.2	4.1	5.5	6.3	7.4	9.7	11
	Indexed ($\times 10^4$)	0.9	1.2	1.6	2.2	2.5	2.9	3.3	3.6	4.0

Table 2. TPC-H: the number of computed and indexed cells when the dimensionality of DIM is fixed to 5.

– SIIC: the Sorted Inverted Index Cube method without pruning;
– SIICP: the Sorted Inverted Index Cube method with pruning;
– DAM: the Dominant Answer Materialization method.

We used BUC [2] to materialize the data cubes for SIIC and SIICP, and the Quotient Cube algorithm [14] to compute the quotient groups for DAM.

5.2. Reducing the Numbers of Aggregate Cells Computed and Indexed

We conducted two experiments to evaluate the effectiveness of reducing the numbers of aggregate cells computed and indexed in our algorithms while the indices were constructed.

For the first set of experiments, we fixed the dimensionality of DIM, and reported the numbers of cells computed and indexed with respect to the increase of dimensionality of UID. We sorted the attributes in the descending order cardinalities. For the TPC-H data set, we generated 9 testing data sets with 2 to 10 dimensions of UID and the dimensionality of DIM was fixed to 5. For the Weather data set, we generated 4 testing data sets with 2 to 5 dimensions if UID and the dimensionality of DIM was fixed to 5.

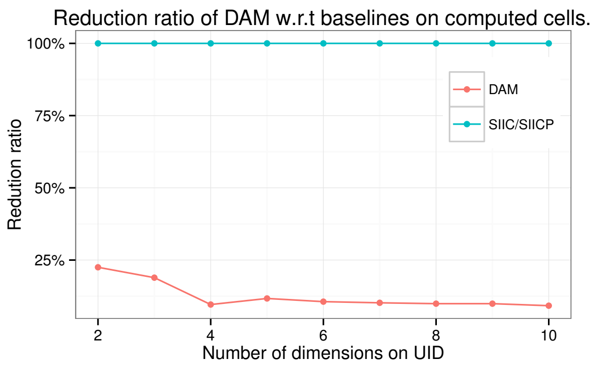
The results are shown in tables 2 and 3. For the materialization step, SIICP and SIIC have the same mechanism; thus, they have the same results for the number of computed and indexed cells. The DAM algorithm shows its advantage over SIICP and SIIC for both the synthetic and the real data sets. Figure 3 shows the reduction ratio of the computed and indexed cells with the TPC-H data set. The reduction ratio is ratio of the number of cells computed by DAM to the number of cells computed by SIICP/SIIC. The reduction ratio in most cases is about 10% meaning that DAM only computes and indices about 10% of the cells that SIICP and SIIC do. Further, the ratio becomes smaller when the dimensionality of UID increases. This indicates that when UID has more dimensions, DAM can save more in materialization and indexing. Figure 4 shows the results with the Weather data set. The trends are similar to those with the synthetic data set.

The savings of DAM are due to the fact DAM only stores and searches the dominant answers in the quotient groups. This mechanism also reduces the runtime and the memory usage in query answering, which will be shown later.

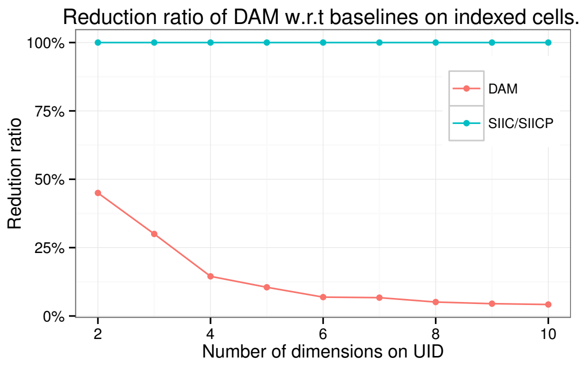
For the second set of experiments, we fixed the dimensionality of UID, and reported the numbers of computed and indexed cells with respect to the dimensionality of DIM. For the TPC-H data set, we generated 4 testing data sets with 2 to 5 dimensions of DIM and the dimensionality of UID was fixed to 10. For the Weather data set, we generated 4 testing data sets with 2 to 5 dimensions of DIM and the dimensionality of UID was fixed to 5.

Method	Dimensionality of UID	2	3	4	5
SIIC / SIICP	Computed ($\times 10^5$)	5.2	11	25	35
	Indexed ($\times 10^4$)	2.1	3.6	5.1	11
DAM	Computed ($\times 10^5$)	0.9	1.5	2.1	2.2
	Indexed ($\times 10^3$)	3.5	4.5	5.1	5.4

Table 3. Weather: the number of computed and indexed cells when the dimensionality of DIM fixed to 5.



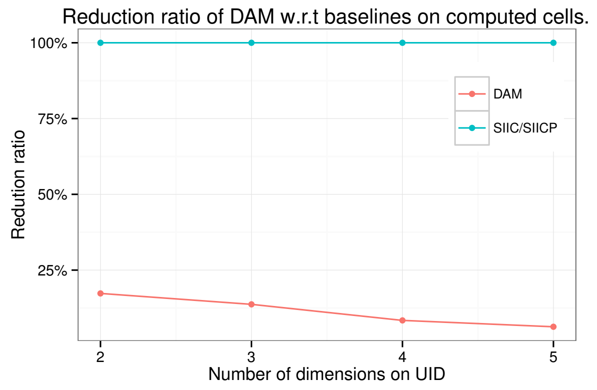
(a) Reduction ratio of computed cells.



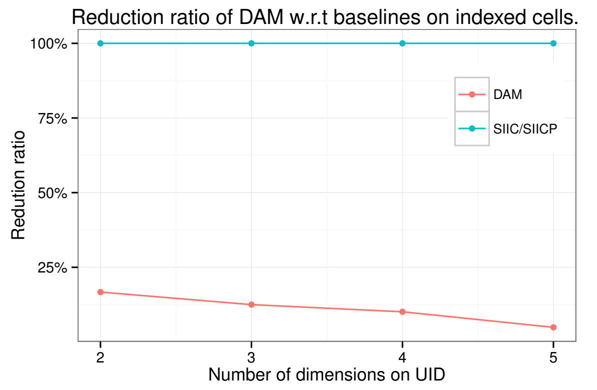
(b) Reduction ratio of indexed cells.

Fig. 3. TPC-H: reduction ratio of DAM to SIIC/SIICP when the dimensionality of DIM is fixed.

The results are shown in tables 4 and 5. Similar to the first set of experiments, SIICP and SIIC have the same mechanism in the materialization step; as such, the two methods have the same numbers of computed and indexed cells. The DAM algorithm significantly outperforms SIICP and SIIC on both with the synthetic and the real data sets. Figure 5 shows the reduction ratio of the computed and indexed cells with the TPC-H data set. The reduction ratio in most cases is under 10%. Similar to the earlier observation, the ratio decreases when the dimensionality of DIM increases. This indicates that when DIM has



(a) Reduction ratio of computed cells.



(b) Reduction ratio of indexed cells.

Fig. 4. Weather: reduction ratio of DAM over SIIC/SIICP when the dimensionality of DIM is fixed.

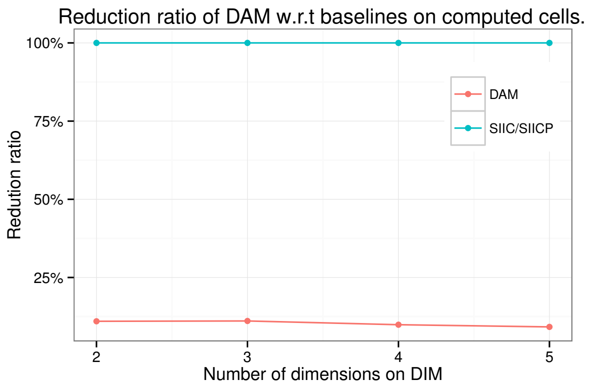
Method	Dimensionality of DIM	2	3	4	5
SIIC / SIICP	Computed ($\times 10^6$)	5.9	7.1	9.6	12
	Indexed ($\times 10^5$)	4.4	6.5	7.9	9.6
DAM	Computed ($\times 10^5$)	6.5	7.9	9.5	11
	Indexed ($\times 10^4$)	2.6	3.1	3.5	4.0

Table 4. TPC-H: the number of computed and indexed cells when the dimensionality of UID is fixed to 10 on the TPC-H data set.

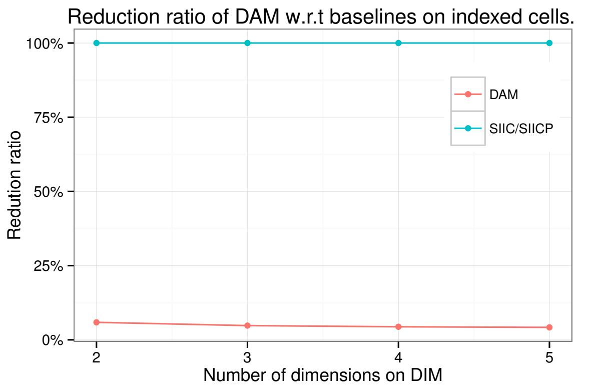
more dimensions, DAM can save more in materialization and indexing. Figure 6 shows the results with the Weather data set and it demonstrates similar trends.

Method	Dimensionality of DIM	2	3	4	5
SIIC / SIICP	Computed ($\times 10^5$)	4.2	9.8	22	35
	Indexed ($\times 10^4$)	1.5	2.6	5.9	11
DAM	Computed ($\times 10^5$)	0.8	1.4	1.9	2.2
	Indexed ($\times 10^3$)	3.1	4.2	4.8	5.4

Table 5. Weather: the number of computed and indexed cells when the dimensionality of UID is fixed to 5 on the Weather data set.



(a) Reduction ratio of computed cells.

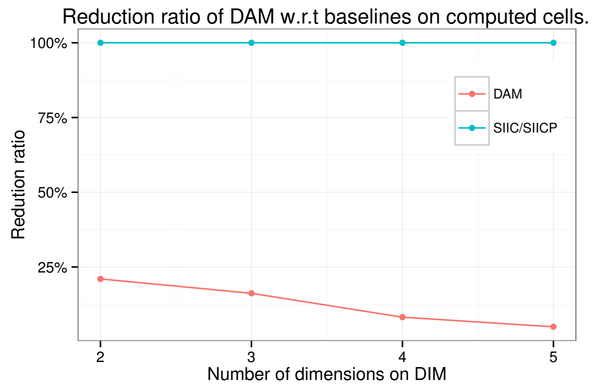


(b) Reduction ratio of indexed cells.

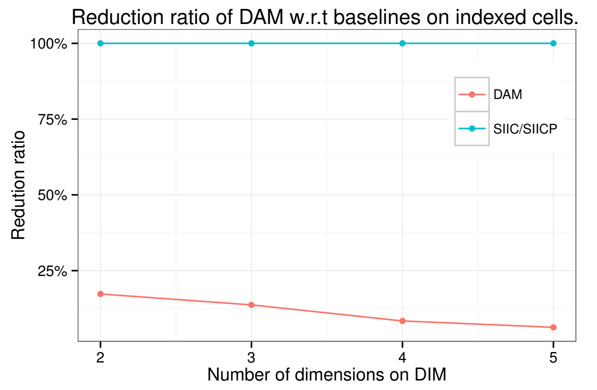
Fig. 5. TPC-H: reduction ratio of DAM to SIIC/SIICP when the dimensionality of UID is fixed.

5.3. Runtime and Memory Usage

We fixed the dimensionality of DIM, and reported both the runtime and the memory usage in index construction and query answering with respect to the dimensionality of UID. The testing data sets were the same as those for the first set of experiments in Section 5.2. The memory usage reported is the peak memory usage in the query answering process. When we tested query answering,



(a) Reduction ratio of computed cells.



(b) Reduction ratio of indexed cells.

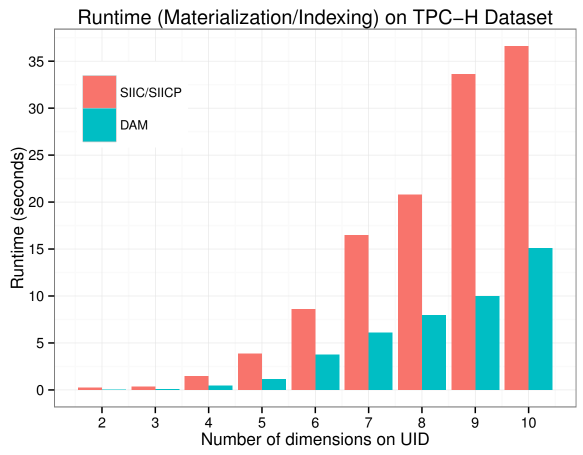
Fig. 6. Weather: reduction ratio of DAM over SIIC/SIICP when the dimensionality of UID is fixed.

100 random queries were conducted; thus, the average query answering runtime and the standard deviation are reported.

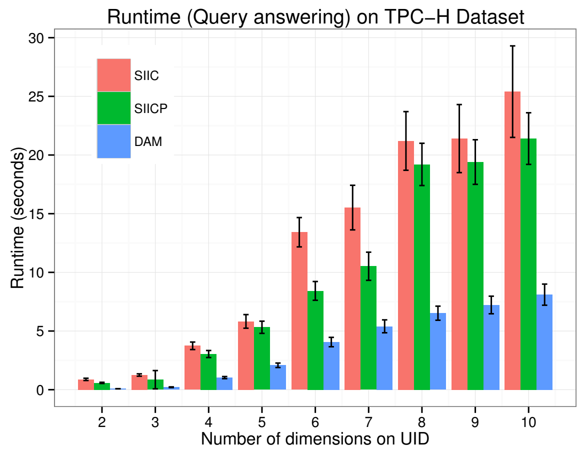
The results with the TPC-H data set and the Weather data set are shown in Figures 7 and 8, respectively.

For the runtime, DAM saves time in both the indexing and the query answering steps. As shown in Figures 7(a) and 8(a), in the index construction step, DAM takes less than half of the runtime of SIIC. Further, the runtime of DAM increases much more slowly than that of SIIC and SIICP when the dimensionality of UID increases. In the index construction step, SIIC and SIICP have the same mechanism; as such, the two methods present the same results for the indexing time. As shown in Figures 7(b) and 8(b), , SIICP is faster than SIIC in the query answering, but is still much more slowly than DAM.

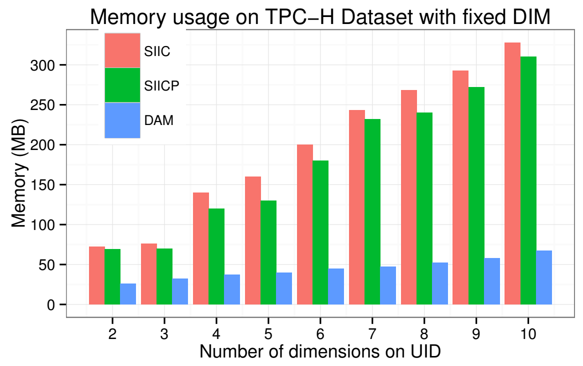
For the memory usage, as shown in Figures 7(c) and 8(c), DAM consumes a small amount of memory, while SIIC and SIICP consume much larger amounts



(a) TPC-H: index construction time with DIM fixed.

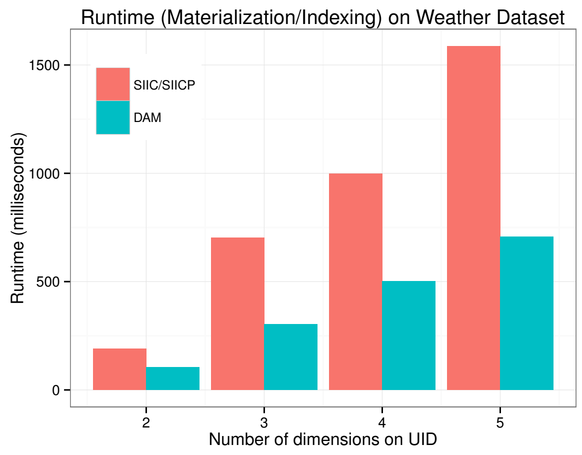


(b) TPC-H: query answering time with DIM fixed.

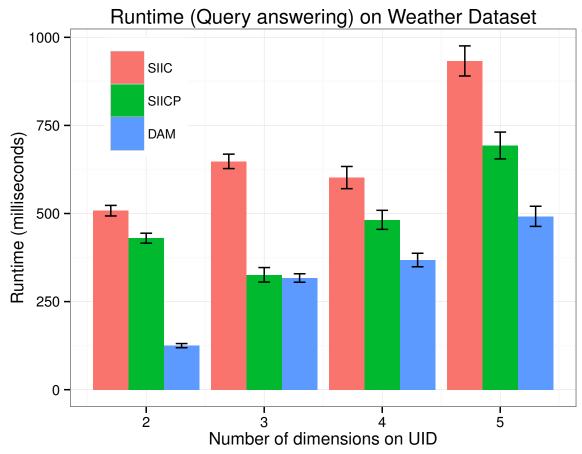


(c) TPC-H: memory usage with DIM fixed.

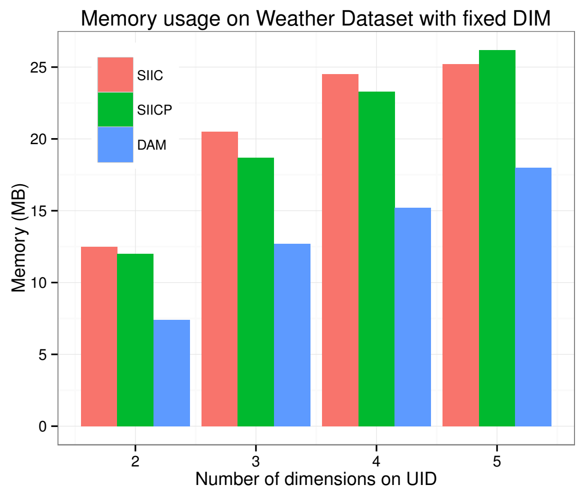
Fig. 7. TPC-H: runtime and memory usage with DIM fixed.



(a) Weather: index construction time with DIM fixed.



(b) Weather: query answering time with DIM fixed.



(c) Weather: memory usage with DIM fixed.

Fig. 8. Weather: runtime and memory usage with DIM fixed.

of memory. Further, the memory usage of DAM increases more slowly than that of SIIC and SIICP when the dimensionality of UID increases. The above results indicate that when UID has more dimensions, DAM can save more time and memory in indexing.

The savings of DAM come from the fact that DAM only computes and stores the dominant answers in the quotient groups. Once a query is given, DAM only searches the dominant answers, which leads to efficiency in both time and memory usage. Both SIIC and SIICP need to materialize the data cube using BUC, and then build the inverted indices. SIICP is faster than SIIC because SIICP applies the pruning techniques in query answering.

Next, we fixed the dimensionality of UID, and reported both the runtime and the memory usage with respect to the dimensionality of DIM. The testing data sets were the same as those for the second set of experiments in Section 5.2.

The results are shown in Figures 9 and 10. The DAM algorithm clearly outperforms SIIC and SIICP on both the real and the synthetic data sets. The runtime and memory usage of DAM increase slower than those of SIIC and SIICP when the dimensionality of DIM increases.

5.4. Scalability

To assess the scalability of our algorithms, we generated and used 4 TPC-H data sets with different sizes: 25%, 50%, 75%, 100% of 1GB using the corresponding TPC-H data set size parameters: 0.25, 0.5, 0.75, 1, respectively. The dimensionality of UID was fixed to 10, and the dimensionality of DIM was fixed to 5. The runtime and the memory usage are reported with respect to the different sizes of the data sets.

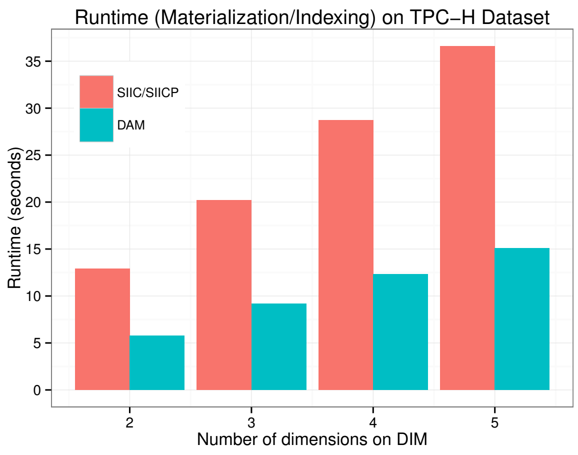
The results are shown in Figure 11. The DAM algorithm is much more scalable than the SIIC and SIICP methods in materialization/indexing as well as query answering. For the memory usage, all the three methods are scalable. DAM consistently uses much less memory than SIIC and SIICP.

6. Conclusions

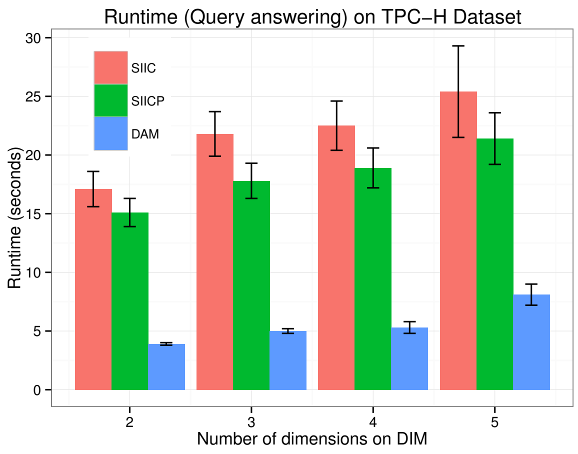
Benchmarking is conducted extensively in various industries for performance improvement. However, performing multidimensional benchmarking efficiently with large data sets remains a technical problem. In this paper, we formulated benchmark queries in the context of data warehousing and business intelligence. Benchmark queries cannot be answered in a straightforward manner using existing OLAP methods. To this end, we developed a few algorithms to answer benchmark queries efficiently.

Our methods employ progressive data cube computation techniques to reduce the number of aggregate cells that need to be computed and indexed. An empirical study using the TPC-H and the Weather data sets demonstrates the efficiency and scalability of our methods. In particular, the DAM method is fast and scalable with large data sets.

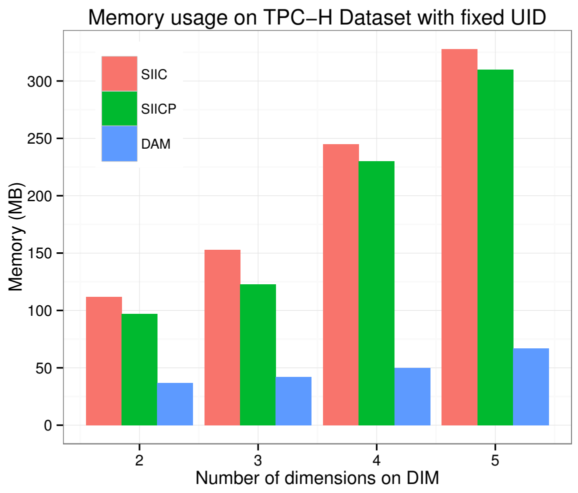
To the best of our knowledge, there are no tools that allow multidimensional benchmarking in data warehouses in business today. We plan to develop tools using the techniques proposed in this paper as our contribution to business at



(a) TPC-H: index construction time with UID fixed.

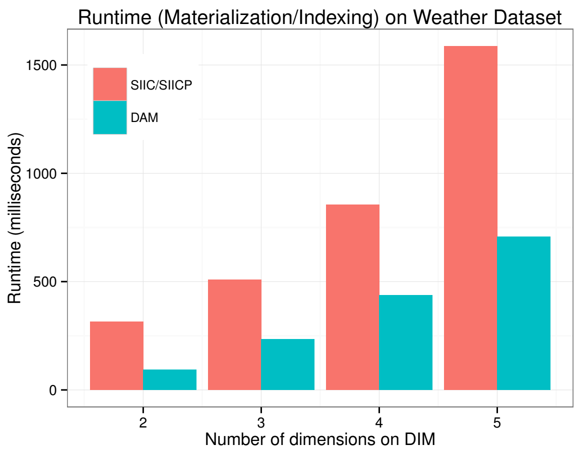


(b) TPC-H: query answering time with UID fixed.

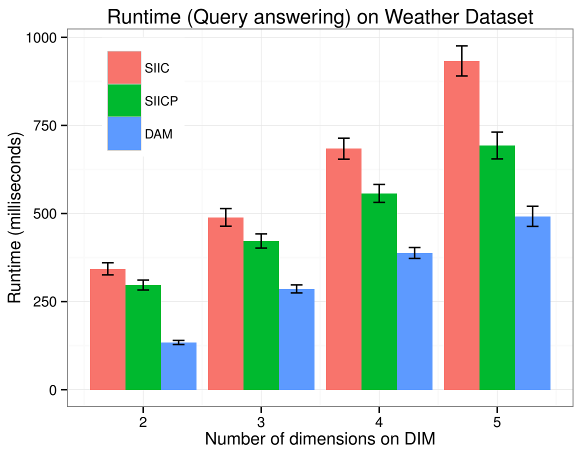


(c) TPC-H: memory usage on TPC-H with UID fixed.

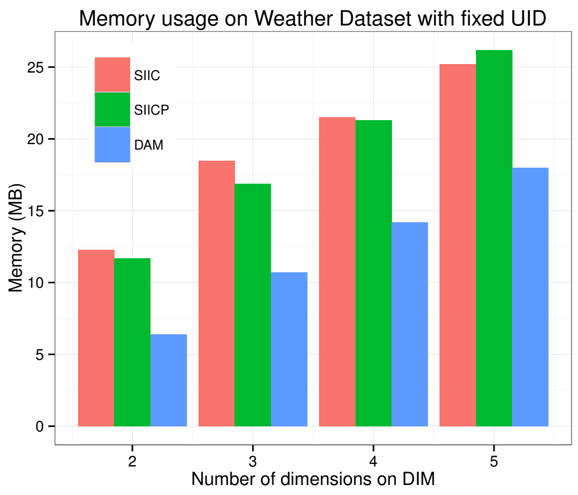
Fig. 9. TPC-H: runtime and memory usage with UID fixed.



(a) Weather: index construction time with UID fixed.

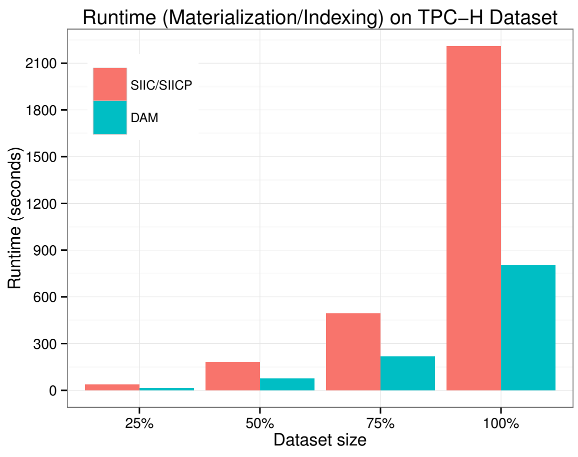


(b) Weather: query answering time with UID fixed.

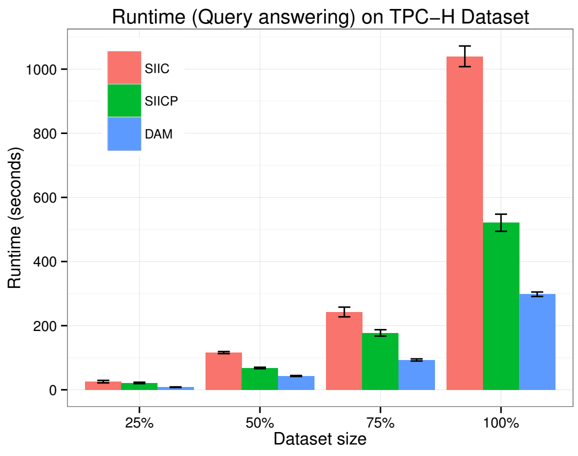


(c) Weather: memory usage with UID fixed.

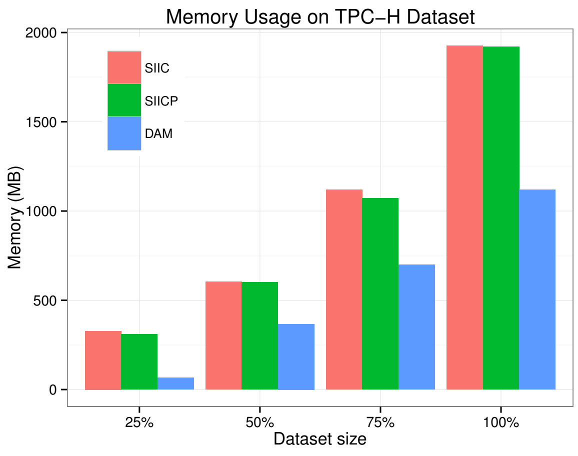
Fig. 10. Weather: runtime and memory usage with UID fixed.



(a) Runtime (Materialization/Indexing) with different data sizes.



(b) Runtime (Query answering) with different data sizes.



(c) Memory usage with different data sizes.

Fig. 11. TPC-H: scalability.

large. We also plan to explore new types of analytic queries and tasks built upon benchmark queries.

7. Acknowledgements

The research was supported in part by an NSERC Discovery Grant and the NSERC CRC Program. All opinions, findings, conclusions and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

References

[1] I. Ajibefun. An evaluation of parametric and non-parametric methods of technical efficiency measurement: Application to small scale food crop production in nigeria. *Journal of agriculture and social sciences*, 4(3):95–100, July 2008.

[2] K. Beyer and R. Ramakrishnan. Bottom-up computation of sparse and iceberg cube. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, SIGMOD '99, pages 359–370, New York, NY, USA, 1999. ACM.

[3] M. Ceci, A. Cuzzocrea, and D. Malerba. Effectively and efficiently supporting roll-up and drill-down OLAP operations over continuous dimensions via hierarchical clustering. *J. Intell. Inf. Syst.*, 44(3):309–333, June 2015.

[4] C. Chen, X. Yan, F. Zhu, J. Han, and P. S. Yu. Graph OLAP: A multi-dimensional framework for graph data analysis. *Knowl. Inf. Syst.*, 21(1):41–63, Oct. 2009.

[5] Y. Chen, F. Dehne, T. Eavis, and A. Rau-Chaplin. Pnp: Sequential, external memory, and parallel iceberg cube computation. *Distrib. Parallel Databases*, 23(2):99–126, Apr. 2008.

[6] G. Dong, J. Han, J. M. W. Lam, J. Pei, and K. Wang. Mining multi-dimensional constrained gradients in data cubes. In *Proceedings of the 27th International Conference on Very Large Data Bases*, VLDB '01, pages 321–330, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[7] M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, and J. D. Ullman. Computing iceberg queries efficiently. In *Proceedings of the 24rd International Conference on Very Large Data Bases*, VLDB '98, pages 299–310, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[8] 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 Min. Knowl. Discov.*, 1(1):29–53, Jan. 1997.

[9] J. Han, J. Pei, G. Dong, and K. Wang. Efficient computation of iceberg cubes with complex measures. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, SIGMOD '01, pages 1–12, New York, NY, USA, 2001. ACM.

[10] Z. He, P. Wong, B. Kao, E. Lo, and R. Cheng. Fast evaluation of iceberg pattern-based aggregate queries. In *Proceedings of the 22Nd ACM International Conference on Conference on Information &#38; Knowledge Management*, CIKM '13, pages 2219–2224, New York, NY, USA, 2013. ACM.

[11] S. Herrera and G. Pang. Efficiency of public spending in developing countries : an efficiency frontier approach. Policy Research Working Paper Series 3645, The World Bank, June 2005.

[12] S. Holder, B. Veronese, P. Metcalfe, F. Mini, S. Carter, and B. Basalisco. Cost benchmarking of air navigation service providers: A stochastic frontier analysis. Technical report, EUROCONTROL, Nov. 2006.

[13] T. Imieliński, L. Khachiyan, and A. Abdulghani. Cubegrades: Generalizing association rules. *Data Min. Knowl. Discov.*, 6(3):219–257, July 2002.

[14] L. V. S. Lakshmanan, J. Pei, and J. Han. Quotient cube: How to summarize the semantics of a data cube. In *Proceedings of the 28th International Conference on Very Large Data Bases*, VLDB '02, pages 778–789. VLDB Endowment, 2002.

[15] E. Lo, B. Kao, W.-S. Ho, S. D. Lee, C. K. Chui, and D. W. Cheung. OLAP on sequence data. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 649–660, New York, NY, USA, 2008. ACM.

[16]R. T. Ng, A. Wagner, and Y. Yin. Iceberg-cube computation with pc clusters. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, SIGMOD '01, pages 25–36, New York, NY, USA, 2001. ACM.

[17]K.-N. T. Nguyen, L. Cerf, M. Plantevit, and J.-F. Boulicaut. Discovering descriptive rules in relational dynamic graphs. *Intell. Data Anal.*, 17(1):49–69, Jan. 2013.

[18]S. Sarawagi, R. Agrawal, and N. Megiddo. Discovery-driven exploration of OLAP data cubes. In *Proceedings of the 6th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '98, pages 168–182, London, UK, UK, 1998. Springer-Verlag.

[19]J. Wang, J. Han, and J. Pei. Closed constrained gradient mining in retail databases. *IEEE Trans. on Knowl. and Data Eng.*, 18(6):764–769, June 2006.

[20]M. Zairi. *Effective Benchmarking.* Springer Netherlands, 1996.