Honey, I Shrunk the Cube

Matteo Golfarelli and Stefano Rizzi

DISI, University of Bologna, Italy
{matteo.golfarelli,stefano.rizzi}@unibo.it

Abstract. Information flooding may occur during an OLAP session when the user drills down her cube up to a very fine-grained level, because the huge number of facts returned makes it very hard to analyze them using a pivot table. To overcome this problem we propose a novel OLAP operation, called shrink, aimed at balancing data precision with data size in cube visualization via pivot tables. The shrink operation fuses slices of similar data and replaces them with a single representative slice, respecting the constraints posed by dimension hierarchies, until the result is smaller than a given threshold. We present a greedy agglomerative clustering algorithm that at each step fuses the two slices yielding the minimum increase in the total approximation, and discuss some experimental results that show its efficiency and effectiveness.

1 Introduction

OLAP is the main paradigm for querying multidimensional databases and data marts. A typical OLAP query asks for returning the values of one or more numerical measures, grouped by a given set of analysis attributes, possibly with reference to a subset of attribute values. The data returned by an OLAP query takes the form of a multidimensional cube, and can be represented either graphically or textually. While graphical representations range from simple line, bar, and pie charts to more sophisticated dashboards involving gauges and geographical maps, the most commonly used textual representation is the so-called *pivot table*. A pivot table usually consists of row, column, and data fields that allow aggregating measures using operators such as sum and average (see Figure 1).

To simplify the querying process for non-ICT users and to support more effectively the decision-making process, OLAP analyses do not normally come in the form of stand-alone queries, but rather of *session*. An OLAP session is a sequence of OLAP queries, each obtained by transforming the previous one through the application of an *OLAP operation*, so as to generate a sort of path that explores relevant areas of the cube. OLAP operations commonly supported by specialized front-end tools are *roll-up*, that further aggregates the data returned by the previous query along dimension hierarchies, *drill-down*, that disaggregates them, and *slice-and-dice*, that selects a subset of values based on some predicate [1].

The effectiveness of OLAP analyses depends on several factors, such as the ability of the front-end tool in intuitively displaying metadata to users and

B. Catania, G. Guerrini, and J. Pokorný (Eds.): ADBIS 2013, LNCS 8133, pp. 176–189, 2013.

City	Year		
	2010	2011	2012
Miami	47	45	50
Orlando	44	43	52
Tampa	39	50	41
Washington	47	45	51
Richmond	43	46	49
Arlington	—	47	52

Fig. 1. A simple pivot table showing data per City and Year

the performance of the underlying multidimensional engine. In particular, when pivot tables are used, one of factors is the achievement of a satisfactory compromise between the precision and the size of the data being visualized. Indeed, as argued in [2], more detail gives more information, but at the risk of missing the overall picture, while focusing on general trends of data may prevent users from observing specific small-scale phenomena. This is also strictly related to the "information flooding" problem, often occurring when a non-expert user is provided with the full expressiveness of OLAP, that may happen because the user drilled down her cube up to a very fine-grained level, where data are very detailed and a huge number of measure values are to be returned [2]. In this case, it is very hard for the user to browse and analyze the resulting data using a pivot table, especially if the device used has limited visualization and data-transmission capabilities (which is becoming more and more common, considering the wide diffusion of smartphones and tablets as supports for individual productivity).

A possible solution to the above-mentioned problem, often adopted in commercial tools, is to put some constraints to the accessible cube exploration paths, e.g., by disallowing drill-downs to the maximum level of data detail (so-called *semi-static reporting*). However, this often leaves users puzzled and unsatisfied. Other approaches devised in the literature that may be used to cope with information flooding are:

- *Query personalization*: when querying, users are allowed to express their *preferences*, e.g., by stating that data for Italian cities are more relevant than those for other cities, or that measure values below a given threshold are the most interesting [3].
- *Intensional query answering*, where the data returned by a query are summarized with a concise description of the properties they share [2].
- *Approximate query answering*, aimed at increasing querying efficiency by quickly returning a concise answer at the price of some imprecision in the returned values [4].
- *OLAM—On-Line Analytical Mining*: the OLAP paradigm is coupled with data mining techniques to create an approach where multidimensional data can be mined "on-the-fly" to extract concise patterns for user's evaluation,

but at the price of an increased computational complexity and an overhead for analyzing the generated patterns [5].

The approach described in this paper can be seen as a form of OLAM based on hierarchical clustering. Starting from the observation that approximation is a key to balance data precision with data size in cube visualization via pivot tables, we propose a novel OLAP operation called *shrink* that can be applied to the cube resulting from an OLAP query to decrease its size while controlling the loss in precision. The shrink operation is ruled by a parameter expressing the maximum size allowed for the resulting data. The idea is to fuse slices of similar data and replace them with a single representative slice (computed as their average), respecting the constraints posed by dimension hierarchies. To this end we present a greedy agglomerative clustering algorithm that at each step fuses the two slices yielding the minimum increase in the total approximation.

The paper outline is as follows. Section 2 introduces a formalization and our working example. Section 3 describes the shrink approach, while Section 4 shows some experimental results. After discussing the related literature in Section 5, in Section 6 we draw the conclusions.

2 Background on Cubes

In this section we introduce a basic formal setting to manipulate multidimensional data. For simplicity we will consider hierarchies without branches, i.e., consisting of chains of attributes, and focus on schemata that include a single measure.

Definition 1 (Multidimensional Schema). *A* multidimensional schema *(or, briefly, a* schema*)* $\mathcal{M}$ *is a couple of*

- *a finite set of disjoint* hierarchies*,* $\{h_1, \ldots, h_n\}$*, each characterized by a set* A_i *of attributes and a* roll-up *total order* $\prec_{h_i}$ *of* A_i*. Each hierarchy attribute* a *is defined over a categorical domain* $Dom(a)$*.*
- *a family of* roll-up *functions that, for each pair of attributes* $a_k, a_j \in A_i$ *such that* $a_k \prec_{h_i} a_j$*, roll-up each value in* $Dom(a_k)$ *to one value in* $Dom(a_j)$*.*

To simplify the notation, we will use letter a for the attributes of h_1, letter b for the attributes of h_2, and so on; besides, we will order the indexes of attributes in each hierarchy according to their roll-up order: $a_1 \prec_{h_1} a_2 \prec_{h_1} \ldots$.

A group-by includes one level for each hierarchy, and defines a possible way to aggregate data.

Definition 2 (Group-by). *A* group-by *of schema* $\mathcal{M}$ *is an element* $G \in A_1 \times \ldots \times A_n$*. A* coordinate *of* $G = \langle a, b, \ldots \rangle$ *is an element* $g \in Dom(a) \times Dom(b) \times \ldots$.

Example 1. IPUMS is a public database storing census microdata for social and economic research [6]. As a working example we will use a simplified form of its CENSUS multidimensional schema based on two hierarchies, namely RESIDENCE and TIME. Within RESIDENCE it is City $\prec_{\text{RESIDENCE}}$ State, and Miami $\in$

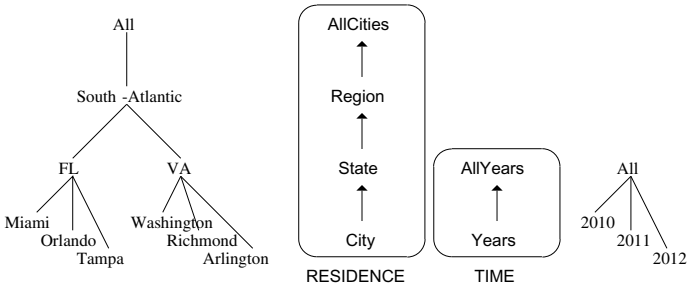

Fig. 2. Roll-up orders and functions for two hierarchies in the CENSUS schema

$Dom(\mathsf{City})$ rolls-up to FL $\in Dom(\mathsf{State})$ (roll-up orders and functions are shown in Figure 2). Some examples of group-by are $G_1 = \langle \mathsf{City}, \mathsf{Year} \rangle$ and $G_2 = \langle \mathsf{State}, \mathsf{AllYears} \rangle$. A coordinate of G_1 is $\langle \text{Miami}, 2012 \rangle$.

An instance of a schema is a set of facts; each fact is characterized by a group-by G that defines its aggregation level, by a coordinate of G, and by a numerical value m. Our shrink operation will be applied to a cube, i.e., to the subset of facts resulting from any OLAP query launched; this can be formalized as follows:

Definition 3 (Cube). *A* cube *at group-by G is a partial function C that maps a coordinate of G to a numerical value (*measure*). Each couple $\langle g, m \rangle$ such that $C(g) = m$ is called a* fact *of C.*

The reason why function C is partial is that cubes are normally *sparse*, i.e., some facts are missing (their measure value is null). An example of missing fact is the one for the Arlington city and year 2010 in Figure 1.

Example 2. Two examples of facts of CENSUS are $\langle \langle \text{Miami}, 2012 \rangle, 50 \rangle$ and $\langle \langle \text{Orlando, } 2011 \rangle, 43 \rangle$. The measure in this case quantifies the average income of citizens. A possible cube at G_1 is depicted in Figure 1.

3 The Shrink Approach

For explanation simplicity we will start by assuming that the shrink operation is applied along hierarchy h_1 to a cube C at the finest group-by, $G = \langle a_1, b_1, \ldots \rangle$.

First of all we observe that, given attribute value $v \in Dom(a_1)$, cube C can be equivalently rewritten as a set of value-slice couples:

$$C = \{\langle v, C^v \rangle, v \in Dom(a_1)\}$$

where C^v is the *slice* of C corresponding to $a_1 = v$ (in the common OLAP sense).

When the shrink operation is applied to C, the slices of C are (completely and disjointly) partitioned into a number of clusters, and all the slices in each cluster are fused into a single, approximate slice, which we call *f-slice*, by averaging their non-null measure values. This means that an f-slice in the shrunk cube

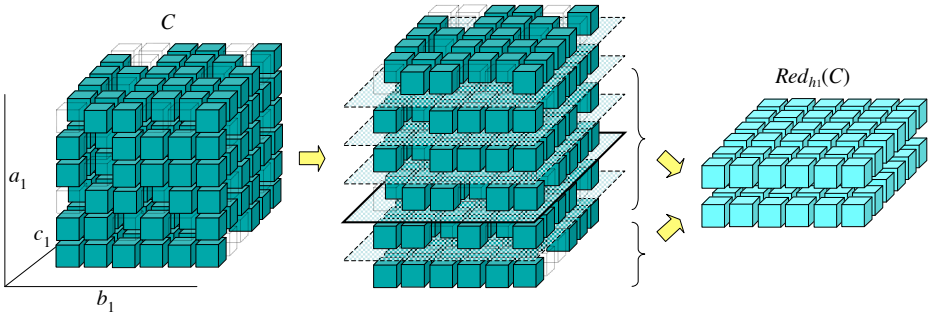

Fig. 3. The shrinking intuition

may not refer to a single value of the domain of a_1, but rather to a set of values —or even to a set of values of a more aggregate attribute $a_2, a_3, \ldots$ in h_1. This process is exemplified in Figure 3, where C is first graphically decomposed into its slices over a_1; these six slices are then partitioned into two clusters including four and two slices, respectively. Finally, the fusion step creates the two f-slices that constitute the shrunk cube.

More precisely, let a_j, $j \geq 1$, be an attribute of h_1, and $V \subseteq Dom(a_j)$ be a set of values of a_j. We denote with $Desc_1(V)$ the set of all values of a_1 that roll-up to a value in V (conventionally, if $j = 1$ it is $Desc_1(V) = V$ for all $V \subseteq Dom(a_1)$). The shrink operation takes cube C in input and returns a *reduction* $Red_{h_1}(C)$ of C, i.e., a set of couples, each formed by a set V of values of any a_j, $j \geq 1$ and by the f-slice F^V that results from fusing the slices in $Desc_1(V)$:

$$Red_{h_1}(C) = \{\langle V, F^V \rangle, V \subseteq Dom(a_j)\}$$

The measure value of each fact in F^V is computed as the average of the non-null measure values of the corresponding facts in the slices belonging to $Desc_1(V)$[1]. F-slice F^V is said to *have level* j; f-slices with different levels can be mixed in a reduction. The *size* of a reduction is the number of f-slices it includes: $size(Red_{h_1}(C)) = |Red_{h_1}(C)|$.

Noticeably, to preserve the semantics of hierarchies in the reduction, the clustering of the slices for fusion must meet some further constraints besides disjointness and completeness:

1. Two slices corresponding to values v and v' of a_1 can be fused in a single f-slice with level j only if both v and v' roll-up to the same value of a_{j+1} (and therefore to the same value of all the subsequent attributes in h_1).
2. If the slices corresponding to all the values of a_1 that roll-up to a single value $\overline{v}$ of a_j ($j > 1$) are *all* fused together, then the corresponding f-slice has level j and is coupled in the reduction with a set V such that $\overline{v} \in V$.

Example 3. Figure 4 shows two possible reductions, with size 3 and 1 respectively, of the cube shown in Figure 1: in (a) the first two rows have been fused

[1] Of course, if $Desc_1(V) = \{v\}$, the measure values in F^V are those in C^v.

City	Year 2010	2011	2012
Miami, Orlando	45.5	44	51
Tampa	39	50	41
Virginia	45	46	50.6

(a)

	Year 2010	2011	2012
South-Atlantic	44	46	49.2

(b)

Fig. 4. Two reductions of the same cube

into a single f-slice (with level 1) referring to {Miami,Orlando} and the last three rows have been replaced by a single f-slice (with level 2) referring to the Virginia state, while in (b) all six rows have been replaced by a single f-slice (with level 3) referring to the South-Atlantic region. Note that, in the f-slice for Virginia shown in Figure 4.a, the income measure has been averaged on three facts for years 2011 and 2012, and on two facts for year 2010 (since the income value for Arlington in 2010 is null in Figure 1). Some examples of slice clustering that would violate the hierarchy semantics are as follows:

$$\{\mathsf{Miami}, \mathsf{Orlando}, \mathsf{Washington}\}, \{\mathsf{Tampa}, \mathsf{Richmond}, \mathsf{Arlington}\} \quad (1)$$

$$\{\mathsf{FL}, \mathsf{Washington}\}, \{\mathsf{Richmond}\}, \{\mathsf{Arlington}\} \quad (2)$$

$$\{\mathsf{FL}\}, \{\mathsf{Tampa}\}, \{\mathsf{Washington}\} \quad (3)$$

(in (1) two slices corresponding to cities of different states are put together; in (2) values of attributes with different levels are mixed; in (3) the clustering is neither complete nor disjoint).

This approach can easily be generalized to be applied to any cube at any group-by $G = \langle a_k, \ldots \rangle$ resulting from an OLAP query; of course, all f-slices in the reduction will have levels not lower than k in this case.

3.1 Measuring the Approximation Error

Fusing some slices into a single f-slice of the reduction when applying the shrink operation gives raise to an approximation. Like in [7], to measure this approximation we use the *sum squared error*.

Given cube C at group-by $G = \langle a_k, b, \ldots \rangle$, let $V \subseteq Dom(a_j)$, $k \leq j$. We denote with $Desc_k(V)$ the set of all values of a_k that roll-up to a value in V, and with F^V the corresponding f-slice. Now let $\overline{g} \in Dom(b) \times Dom(c) \times \ldots$ be an incomplete coordinate of G (no value for attribute a_k is given). The *sum squared error* (SSE) associated to F^V is

$$SSE(F^V) = \sum_{\overline{g} \in Dom(b) \times Dom(c) \times \ldots} \; \sum_{v \in Desc_k(V)} (C^v(\overline{g}) - F^V(\overline{g}))^2 \quad (4)$$

(conventionally, $C^v(\overline{g}) - F^V(\overline{g}) = 0$ if $C^v(\overline{g})$ is null). Given reduction $Red_{h_1}(C)$, the SSE associated to $Red_{h_1}(C)$ is

$$SSE(Red_{h_1}(C)) = \sum_{\langle V,F^V\rangle \in Red_{h_1}(C)} SSE(F^V) \tag{5}$$

Example 4. The SSEs associated to the f-slices in Figure 4.a are $(2.25 + 2.25) + (1 + 1) + (1 + 1) = 8.5$, 0, and 14.68 respectively; the overall SSE associated to the reduction is 23.2. The SSE associated to the reduction in Figure 4.b is 158.8.

3.2 A Heuristic Algorithm for the Shrink Operation

Given a cube C, a combinatorial number of possible reductions can be operated on it, one for each way of clustering the slices of C by preserving the hierarchy semantics. Of course, the more the reduction process is pushed further, the lower the number of resulting f-slices; hence, the lower the size of the data returned to the user but the higher the approximation introduced. So, it is apparent that the reduction process should be driven by a parameter $size_{max}$ expressing the trade-off between size and precision, in particular the maximum tolerable number of f-slices in the reduction (determined for instance by the size of the display and/or by the network bandwidth of the device). In the light of this, the problem of finding a reduction of C along h_1 can be so formulated:

Problem 1 (Reduction Problem). Find the reduction that yields the minimum SSE among those whose size is not larger than $size_{max}$.

The reduction problem has exponential complexity, which is hardly compatible with the inherent interactivity of OLAP sessions. Indeed, the presence of hierarchy-related constraints reduces the problem search space, so the worst case for the reduction problem is the one where no such constraints are present (i.e., all values of a_k roll-up to the same value of a_{k+1}). In this case, the size of the search space is given by the $|Dom(a_k)|$-th Bell number[2], i.e., the number of different partitions of a set with $|Dom(a_k)|$ elements [8]. So, there is a need for a heuristic approach that satisfies real-time computational feasibility while preserving the quality of the solutions obtained. In this direction, we observe that a reduction is determined starting from a clustering of the slices in C, where each cluster determines an f-slice and the size of the reduction is given by the number of clusters. Then, we show how the reduction problem can be solved in a sub-optimal way by applying an agglomerative algorithm for hierarchical clustering with constraints to the set of slices in C.

Hierarchical clustering aims at building a hierarchy of clusters [9]; this can be done following either a top-down or a bottom-up approach. The algorithms based on a bottom-up approach are called *agglomerative*: each element (each slice of C, in our case) initially stands in its own cluster, then pairs of clusters are progressively merged. The decision of which pair of clusters will be merged is

[2] The Bell number is defined as follows: $B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k$, $B_0 = B_1 = 1$.

Algorithm 1. The Shrinking Algorithm

Require: C at group-by $G = \langle a_k, b, \ldots \rangle$, $size_{max}$
Ensure: $Red_{h_1}(C)$
1: $Red_{h_1}(C) \leftarrow \emptyset$ ▷ Initialize reduction...
2: **for all** $v \in Dom(a_k)$ **do** $Red_{h_1}(C) \leftarrow Red_{h_1}(C) \cup \{\langle \{v\}, C^{\{v\}} \rangle\}$ ▷ ...one slice per f-slice
3: **while** $size(Red_{h_1}(C)) > size_{max}$ **do** ▷ Check constraint on maximum size
4: find $F^{V'}, F^{V''} \in Red_{h_1}(C)$ s.t. $F^{V'}$ and $F^{V''}$ have the same level j
5: **and** $SSE(F^{V' \cup V''}) - SSE(F^{V'}) - SSE(F^{V''})$ is minimal
6: **and** all values in $V' \cup V''$ roll-up to the same value of a_{j+1} ▷ Hierarchy constraints
7: $Red_{h_1}(C) \leftarrow Red_{h_1}(C) \setminus \{F^{V'}, F^{V''}\}$ ▷ Merge into an f-slice...
8: **if** $\exists \overline{v} \in Dom(a_{j+1})$ s.t. $Desc_j(\overline{v}) = V' \cup V''$ **then**
9: $Red_{h_1}(C) \leftarrow Red_{h_1}(C) \cup \{F^{\{\overline{v}\}}\}$ ▷ ...either at level $j+1$
10: **else**
11: $Red_{h_1}(C) \leftarrow Red_{h_1}(C) \cup \{F^{V' \cup V''}\}$ ▷ ...or at level j
12: **return** $Red_{h_1}(C)$

usually taken in a greedy (i.e., locally optimal) manner. In our context, merging two clusters means merging two f-slices (i.e., fusing all the slices belonging to each of the two f-slices into a single f-slice). As a merging criterion we adopted the *Ward's minimum variance method* [9], i.e., at each step we merge the pair of f-slices that leads to minimum increase ΔSSE in the total SSE of the corresponding reduction. Of course, two f-slices can be merged only if the resulting reduction preserves the hierarchy semantics as explained above. Finally, the agglomerative process is stopped when the next merge would violate the constraint expressed by $size_{max}$. The overall process is outlined in Algorithm 1; remarkably, since the SSE grows monotonically at each merge, by simply changing line 3 the same algorithm can be used to solve a symmetrical formulation of the reduction problem asking for the reduction that has minimum size among those whose SSE is below a threshold.

Interestingly, it can be proven that the SSE of a reduction can be incrementally computed, i.e., the SSE of an f-slice $F^{V' \cup V''}$ obtained by merging two f-slices $F^{V'}$ and $F^{V''}$ (line 5) can be computed from the SSEs of the f-slices to be merged as follows:

$$SSE(F^{V' \cup V''}) = SSE(F^{V'}) + SSE(F^{V''}) + \times \sum_{\overline{g} \in Dom(b) \times Dom(c) \times \ldots} \frac{H'_{\overline{g}} \cdot H''_{\overline{g}}}{H'_{\overline{g}} + H''_{\overline{g}}} \cdot (F^{V'}(\overline{g}) - F^{V''}(\overline{g}))^2 \quad (6)$$

where $H'_{\overline{g}} = |\{v \in Desc_k(V') \text{ s.t. } C^v(\overline{g}) \text{ is not null}\}|$ (similarly for $H''_{\overline{g}}$).

Example 5. Consider again the cube in Figure 1. In the following we show in detail how Algorithm 1 computes a reduction with $size_{max} = 3$ (Figure 5).

1. In the initialization step (line 2), six singleton f-slices are created, one for each slice of C. Since this first reduction has size 6, it violates the $size_{max}$ constraint and the *while* cycle is entered.
2. The most promising merge is the one between the Arlington and the Washington slices, that yields $\Delta SSE = 2.5$ (Figure 5.a, right).

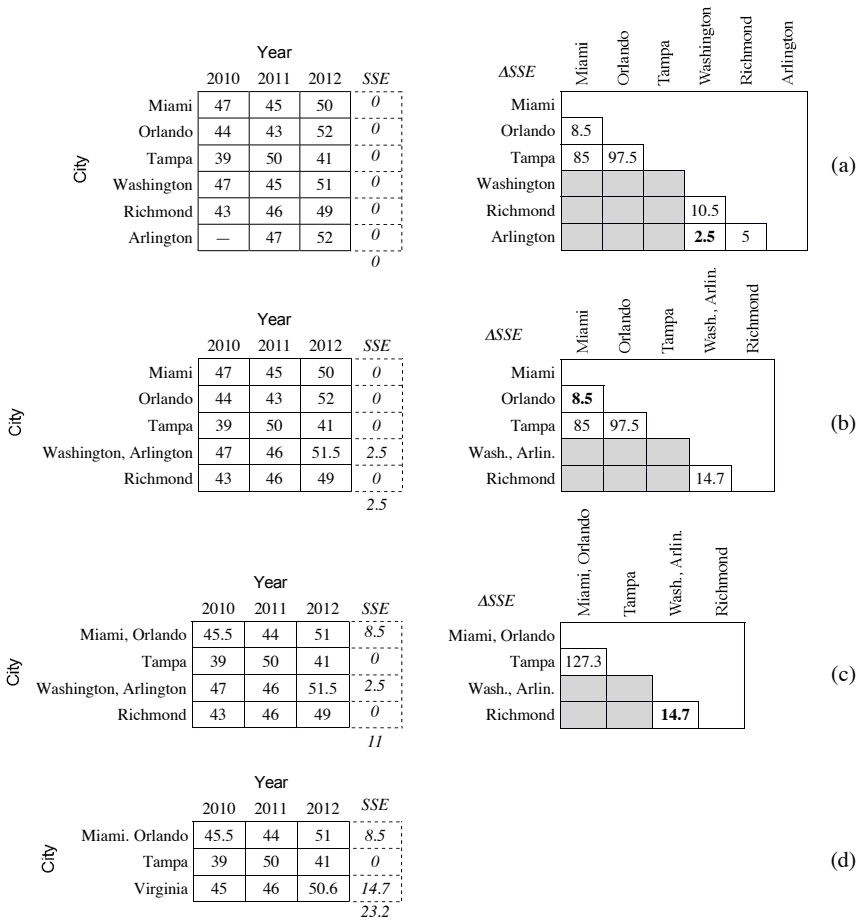

Fig. 5. Applying the agglomerative clustering algorithm

3. In the resulting reduction (Figure 5.b, left), that still violates the $size_{max}$ constraint, the most promising merge is the one between the Miami and the Orlando slices (Figure 5.b, right).
4. At this step, the Richmond slice is fused with the Washington-Arlington f-slice (Figure 5.c, right). Since the resulting f-slice covers all the Virginia state, its level is changed to 2 (Figure 5.d).
5. The resulting reduction meets the $size_{max}$ constraint, so the algorithm stops.

The worst-case complexity of Algorithm 1 in terms of operations for computing ΔSSE refers to the case where no hierarchy-related constraints are present and $size_{max} = 2$, in which case it is $O(\delta^3 \cdot |C^v|)$ where $\delta = |Dom(a_k)|$ and $|C^v|$ is the size of each slice. Specifically, each ΔSSE computation requires $|C^v|$ operations according to Equation 6 (because it is $|Dom(b)| \cdot |Dom(c)| \cdot \ldots = |C^v|$); if $size_{max} = 2$ the while loop (line 3 of Algorithm 1) is executed $\delta - 2$ times, and

Table 1. The four cubes used for testing

Cube	# facts	Sparsity	# facts per slice
C_0	34,008,000	0.6%	21,800
C_1	13,603,200	11.4%	8,720
C_2	1,622,400	4.5%	1,040
C_3	28,080	22.2%	18

it requires $\frac{(\delta-i+1)(\delta-i)}{2}$ ΔSSE computations (line 5) at the i-th time, yielding a total of $\frac{\delta(\delta^2-2)}{6} - 1$ computations.

4 Experimental Results

This section collects the main results of the tests we carried out to evaluate the behavior of the shrink operation. The shrink algorithm is implemented in C++ and has run on a Pentium i5 quad-core (2.67GHz, 4 GB RAM, Windows 7-64 bit). The experiments have been carried out on four 4-dimensional cubes extracted from the IPUMS database, whose properties are summarized in Table 1. The cubes from C_1 to C_3 are obtained by grouping the facts in C_0 at progressively coarser granularities.

For each cube, we applied the shrink operation to the City attribute of the RESIDENCE hierarchy. Figure 6 shows the SSE and the corresponding number of non-null facts in the reduction for decreasing values of $size_{max}$ from 1560 (no shrinking, all city slices are kept distinct) to 1 (all city slices are fused into a single f-slice). As expected, the SSE shows an exponential behavior that is more apparent for cubes at coarser group-by's. The decrease in the number of non-null facts is also related to the features of the cube to which shrinking is applied, namely, their sparsity and their distribution of measure values. More details on these issues follow:

- When a non-null fact is fused with a null one, the contribution to the SSE is 0 (see Section 3.1). Thus, the SSE increases slowly in cubes with a fine group-by due to their high sparsity.
- The SSE increases slowly for high values of $size_{max}$ because Algorithm 1 merges the f-slices with lowest SSE first and because sparsity decreases as the shrinking process is pushed further.
- While the decrease in the total number of facts of a reduction strictly depends on the size of the slices to be fused (when two slices are fused, the total number of facts is always reduced by the total number of facts in a slice), the decrease of non-null facts varies according to the sparsity.
- Sparsity is not the only factor that determines the SSE values and trend. Facts at a specific group-by can be characterized by measure values with particularly high variance. For instance, this is the case for C_2, which is sparser than C_3 and has a finer group-by, but initially leads to higher SSEs.

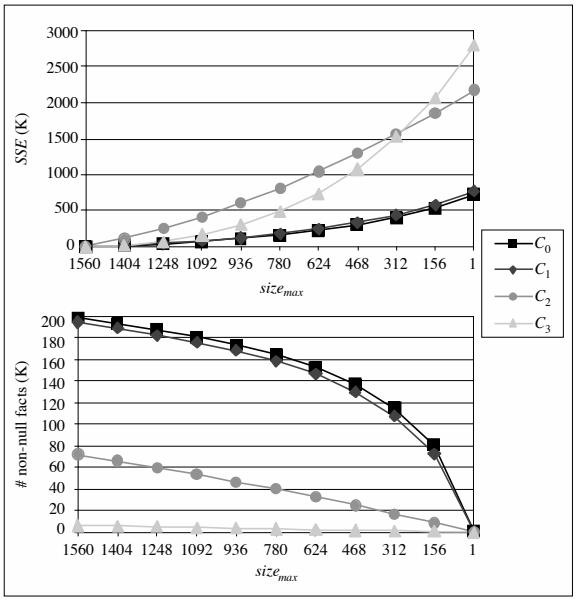

Fig. 6. SSE (top) and number of non-null facts in the reduction (bottom) for decreasing values of $size_{max}$

The overall execution time for reducing our largest cube, C_0, to two f-slices ($size_{max} = 2$) is 3.08 secs, with an average duration of each reduction step of about 2 milliseconds. The time is not significantly smaller for the other cubes, because its major component is the search of the minimum SSE, which only depends on the number of f-slices to be compared (due to the enforcement of hierarchy-related constraints, only a small number of SSE computations are required, so the size of each slice is not very influential).

To better evaluate the trade-off between sub-optimality and time-saving introduced by our heuristics, we also implemented an optimal algorithm based on a branch-and-bound approach that carries out an exhaustive enumeration of the feasible solutions. The enumeration technique we adopted was initially proposed in [8] for enumerating in a systematic manner all the partitions of the leafs of a generalization hierarchy. A node in the search space, corresponding to a possible partition, is cut when its SSE is greater or equal to the one of the best feasible solution found so far; the initial upper-bound was set to our heuristic solution. Among the different criteria for choosing the next node to be expanded (i.e., the two f-slices to be merged), the most effective one turned out to be a best-first search choosing to merge the two f-slices for which the ratio of the SSE obtained and the relative reduction in cube size is minimum. Unfortunately, the huge size of the search space allowed us to solve to optimality only very small problems ($|Dom(a_k)| \leq 23$ and $size_{max} = 0.3 \cdot |Dom(a_k)|$. In all the experiments, the gap between the optimal and heuristic solution in terms of SSE value turned out to be less than 10%. The execution time of the optimal algorithm was about

30 mins when $|Dom(a_k)| = 23$; when $|Dom(a_k)| = 55$, no optimal solution was found after about 1 day.

5 Related Literature

Many studies in the literature can be related to the information flooding problem.

Query personalization allows a reduction in the output size by providing a mechanism for selecting only the information relevant to the users. In [3] the authors propose an algebra for defining preferences in the OLAP context. This approach is powerful from an expressiveness point of view, but differently from the shrink operation, it requires users to manually specify their preference criteria. In [10] a technique is proposed for obtaining a personalized visualization of OLAP query results in presence of constraints on the maximum number of returned cells for each dimension of analysis; user preferences are manually defined in terms of a total order of the dimensional values. Also most other works on OLAP content personalization, such as [11], share with the two mentioned above the need for some manual intervention to define preferences. In a few cases, such as [12], preferences are automatically derived by analyzing the log of the user queries, which makes the results of current queries dependent on the past queries —while the results of shrinking are always independent of the past.

According to [13], an *intensional answer* to a query, instead of returning a precise and complete set of tuples, summarizes them with a concise description of the properties these tuples share. Intensional query answering has been applied in many area of computer science (e.g., object-oriented databases [14], deductive database [15], and question answering systems [16,17]) but, to the best of our knowledge, the only work related to the OLAP area is [2], that proposes a framework for computing an intensional answer to an OLAP query by leveraging the previous queries in the current session. The idea is to use an intensional answer to concisely characterize the cube regions whose data do not change along the sequence, coupled with an extensional answer to describe in detail only the cube regions whose data significantly differ from the expectation. Like for [12], even here query results strongly depend on the user history.

Another research topic that is related to our is *approximate query answering*, whose main goal is to increase query efficiency by returning a reduced result while minimizing the approximation introduced —which is clearly similar to our goal. Some approximate query answering approaches were specifically devised for OLAP. For instance, in [18] the authors propose a set of techniques that, given a fixed amount of space, return the cells that maximize the accuracy for all possible group-by queries on a set of columns. While the shrink operation uses approximation to reduce the size of query results, in [18] using sampling to quickly compute measure values for each group introduces an approximation but does not change the size of results. In [19] the focus is different: tuples are sent to a data warehouse as separate streams that must be integrated, and the approximation of queries when a small quantity of memory is available is studied. OLAP query approximation was also studied in the context of data

exchange [20]. In particular, in [19] a data warehouse is fed by several sources and the authors study how to sample each source to guarantee that the union of the samples correctly approximates the union of the sources. [21] considers a peer-to-peer data warehouse and studies how to approximately answer OLAP queries by mapping the information available on a source peer on the schema of a target peer.

Finally, an approach similar to ours in the area of temporal databases is *parsimonious temporal aggregation* (PTA), a novel temporal aggregation operator that merges tuples related to the same subject and with consecutive time intervals to computes aggregation summaries that reflect the most significant changes in the data over time [22]. Though our technique shares the same basic principle, it bears more complexity because (a) the constraints deriving from temporal consecutiveness strongly reduce the PTA search space; (b) preserving hierarchy semantics introduces additional complexity; (c) PTA computes the approximation level on a single numerical attribute while we work with complex multidimensional slices.

6 Conclusions and Future Works

In this paper we have proposed a novel OLAP operation, called shrink, to cope with the information flooding problem by enabling users to balance the size of query results with their approximation. We described a heuristic implementation of the shrink operation and discussed its efficiency and effectiveness. Remarkably, since the proposed algorithm works by progressively merging couples of f-slices, when using the shrink operation during an OLAP session the shrunk results obtained for a given size threshold can always be reused to compute more efficiently the results for a lower size threshold.

To enhance the shrink approach, we are currently working on two relevant issues. Firstly, to improve effectiveness, we will extend the formulation of the operation to work on several hierarchies simultaneously. This will obviously make the problem much harder from the computational point of view, thus bringing to the forefront efficiency issues. To cope with this, we will study smarter heuristics to obtain good solutions through a greedy approach on the one hand, specific optimization techniques to obtain optimal solution through dynamic programming approaches on the other.

References

1. Golfarelli, M., Rizzi, S.: Data Warehouse design: Modern principles and methodologies. McGraw-Hill (2009)
2. Marcel, P., Missaoui, R., Rizzi, S.: Towards intensional answers to OLAP queries for analytical sessions. In: Proc. DOLAP, Maui, USA, pp. 49–56 (2012)
3. Golfarelli, M., Rizzi, S., Biondi, P.: myOLAP: An approach to express and evaluate OLAP preferences. IEEE Trans. Knowl. Data Eng. 23(7), 1050–1064 (2011)

4. Vitter, J.S., Wang, M.: Approximate computation of multidimensional aggregates of sparse data using wavelets. In: Proc. SIGMOD, Philadelphia, USA, pp. 193–204 (1999)
5. Han, J.: OLAP mining: Integration of OLAP with data mining. In: Proc. Working Conf. on Database Semantics, Leysin, Switzerland, pp. 3–20 (1997)
6. Minnesota Population Center: Integrated public use microdata series (2008), http://www.ipums.org
7. Gordevicius, J., Gamper, J., Böhlen, M.H.: Parsimonious temporal aggregation. VLDB J. 21(3), 309–332 (2012)
8. Li, T., Li, N.: Towards optimal k-anonymization. DKE 65(1), 22–39 (2008)
9. Tan, P.N., Steinbach, M., Kumar, V.: Introduction to Data Mining. Pearson International (2006)
10. Bellatreche, L., Giacometti, A., Marcel, P., Mouloudi, H., Laurent, D.: A personalization framework for OLAP queries. In: Proc. DOLAP, Bremen, Germany, pp. 9–18 (2005)
11. Jerbi, H., Ravat, F., Teste, O., Zurfluh, G.: A framework for OLAP content personalization. In: Catania, B., Ivanović, M., Thalheim, B. (eds.) ADBIS 2010. LNCS, vol. 6295, pp. 262–277. Springer, Heidelberg (2010)
12. Aligon, J., Golfarelli, M., Marcel, P., Rizzi, S., Turricchia, E.: Mining preferences from OLAP query logs for proactive personalization. In: Eder, J., Bielikova, M., Tjoa, A.M. (eds.) ADBIS 2011. LNCS, vol. 6909, pp. 84–97. Springer, Heidelberg (2011)
13. Motro, A.: Intensional answers to database queries. IEEE Trans. Knowl. Data Eng. 6(3), 444–454 (1994)
14. Yoon, S.C., Song, I.Y., Park, E.K.: Intelligent query answering in deductive and object-oriented databases. In: Proc. CIKM, Gaithersburg, USA, pp. 244–251 (1994)
15. Flach, P.: From extensional to intensional knowledge: Inductive logic programming techniques and their application to deductive databases. Technical report, University of Bristol, Bristol, UK (1998)
16. Benamara, F.: Generating intensional answers in intelligent question answering systems. In: Proc. Int. Conf. Natural Language Generation, Brockenhurst, UK, pp. 11–20 (2004)
17. Cimiano, P., Rudolph, S., Hartfiel, H.: Computing intensional answers to questions – an inductive logic programming approach. DKE 69(3), 261–278 (2010)
18. Acharya, S., Gibbons, P.B., Poosala, V.: Congressional samples for approximate answering of group-by queries. In: Proc. SIGMOD Conference, Dallas, USA, pp. 487–498 (2000)
19. de Rougemont, M., Cao, P.T.: Approximate answers to OLAP queries on streaming data warehouses. In: Proc. DOLAP, Maui, USA, pp. 121–128 (2012)
20. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data exchange: Semantics and query answering. In: Calvanese, D., Lenzerini, M., Motwani, R. (eds.) ICDT 2003. LNCS, vol. 2572, pp. 207–224. Springer, Heidelberg (2002)
21. Golfarelli, M., Mandreoli, F., Penzo, W., Rizzi, S., Turricchia, E.: OLAP query reformulation in peer-to-peer data warehousing. Inf. Syst. 37(5), 393–411 (2012)
22. Gordevicius, J., Gamper, J., Böhlen, M.H.: Parsimonious temporal aggregation. VLDB J. 21(3), 309–332 (2012)