Materialization of fragmented views in multidimensional databases

Matteo Golfarelli [a], Vittorio Maniezzo [b], Stefano Rizzi [a,*]

[a] *DEIS, University of Bologna, Viale Risorgimento 2, 40136 Bologna, Italy*
[b] *Department of Computer Science, University of Bologna, Mura Anteo Zamboni 7, 40127 Bologna Italy*

Abstract

The most effective technique to enhance performances of multidimensional databases consists in materializing redundant aggregates called views. In the classical approach to materialization, each view includes all and only the measures of the cube it aggregates. In this paper we investigate the benefits of materializing views in vertical fragments, aimed at minimizing the workload response time. We formalize the fragmentation problem as a 0–1 integer linear programming problem, which is then solved by means of a standard integer programming solver to determine the optimal fragmentation for a given workload. Finally, we demonstrate the usefulness of fragmentation by presenting a large set of experimental results based on the TPC-H benchmark.
© 2003 Elsevier B.V. All rights reserved.

1. Introduction

Recently, multidimensional databases (MDs) have gathered wide research and market interest as the core of decision support applications such as data warehouses. A MD can be seen as a collection of multidimensional *cubes* centered on facts of interest (for instance, the line items in the orders issued by a company). A cube is composed by *cells* which contain a set of numerical

---

* Corresponding author. Tel.: +39-51-2093542; fax: +39-51-2093540.
E-mail addresses: mgolfarelli@deis.unibo.it (M. Golfarelli), maniezzo@csr.unibo.it (V. Maniezzo), srizzi@deis.unibo.it (S. Rizzi).

measures conveying useful information for the decision process (for instance, the price and the quantity of a line item). The cube *dimensions* are attributes which represent possible dimensions for analysis (for instance, products, suppliers and orders) [1]. From the logical point of view, the set of dimensions functionally determines every measure. Furthermore, each dimension may be associated to a hierarchy of *attributes* which describe it (for instance, the city and the nation of a supplier).

An MD implemented on a relational DBMS is usually organized according to the so-called *star scheme* [2], in which each cube is represented by one *fact table* storing the measures and one denormalized *dimension table* for each dimension of analysis. The primary key of the fact table is a set of foreign keys, each referencing the (usually surrogate) key of a dimension table. Besides the key, each dimension table contains all the attributes which describe the corresponding dimension.

In MDs, the response to the workload is improved by materializing a set of *views* which store the pre-computed results to frequent/critical queries, thus introducing some degree of data redundancy. Since the basic mechanism to extract useful information from elemental data in MDs is aggregation, a view on a cube is typically defined by a SQL query which groups data by a set of attributes taken from dimension tables (*grouping set*) and computes summarized values for measures by means of some aggregation operators (see Fig. 1). Even views are organized into star schemes. In the following, with a slight abuse in terminology, we will call views *all* the star schemes for a cube: the one storing elemental data (*primary view*) and those containing aggregated values (*secondary views*).

Since pre-computing and storing all the possible secondary views is infeasible, several techniques have been proposed to select the subset to materialize in order to optimize the response to the workload (e.g., [3–6]). In these approaches, each view includes the attributes in the grouping set and *all and only* the measures of a single cube, without considering which measures are required *together* by the queries in the workload; we will globally refer to them as *classic* approaches.
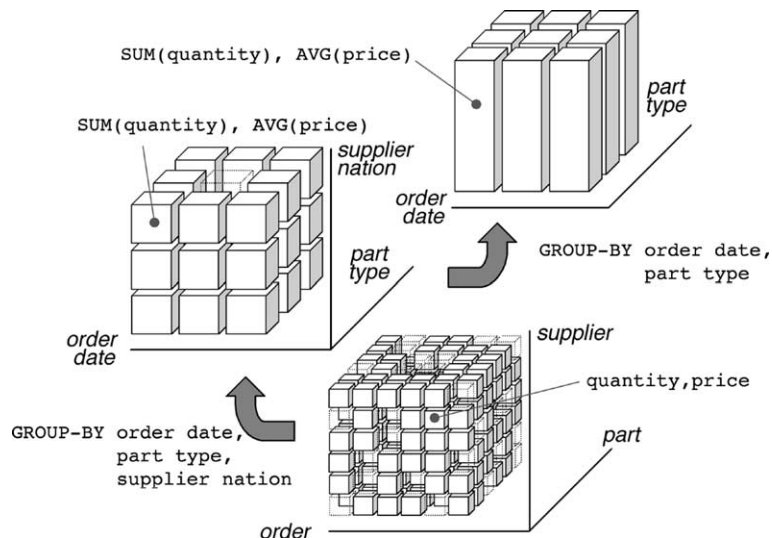

Fig. 1. Two views on the line item cube.

In this paper we investigate how the response to the workload can be enhanced by materializing views in *vertical fragments*, each including a subset of measures possibly taken from different cubes, aggregated on the same grouping set. Thus, as compared to classic materialization, our approach may achieve the goal of unifying two or more views into a single fragment as well as that of partitioning a view into two or more fragments. The most evident motivations for applying vertical fragmentation techniques to views in MDs can be stated as follows:

- As compared to operational databases, in MDs the benefits of fragmentation are further enhanced by the multiple query execution plans due to the presence of redundant secondary views. These benefits are particularly relevant if the MD is implemented on a parallel architecture; if disk arrays are adopted and fragmentation is coupled with an allocation algorithm, the queries requiring multiple fragments allocated on different disks can be effectively parallelized [7,8].
- While every measure of each cube must be stored in its elemental form to avoid loss of information, the same is not true for aggregated values. Thus, depending on the workload, some measures may be not materialized at all in aggregated fragments, leading to space saving as compared to classic materialization. When a space constraint is posed, this saving can be profitably employed to store other useful fragments.
- Though partitioning a view entails replicating its key, fragmentation entails no significant storage overhead since surrogate keys require few bytes to be stored and the number of measures typically exceeds the number of dimensions (especially in secondary views). Furthermore, storing in the same fragment measures taken from different cubes leads to saving space.

Vertical fragmentation for MDs has already been proposed in the literature [7,9,10]; in this paper we give the following new contributions:

- *Optimality*. In the other approaches, no suggestions on how to determine a good fragmentation were given. In this paper, the fragmentation problem is formalized as a 0–1 integer programming (IP) problem, which is then solved by means of a standard IP solver. Given a core workload, a constraint on the global disk space for materialization and a function to estimate the cost for executing each query on each fragment, an *optimal* fragmentation is thus produced.
- *Generality*. Our approach subsumes the classic one: in fact, if every query in the workload requires all and only the measures of a single cube, all the views obtained are unfragmented.
- *Testing*. The results of a large set of experiments is reported in the paper in order to quantify and discuss the benefits of fragmentation from different points of view.

In particular, with reference to previous contributions [10,11], this paper introduces a new mathematical formulation of the vertical fragmentation problem and exploits its potential by directly applying an efficient IP solver (namely CPLEX, [12]) to the algebraic translation of the problem elements. In this way it is possible to solve to optimality real-world sized instances much larger than those solved in [10]: there, in fact, a branch-and-bound approach was adopted, and time constraints forced to pre-maturely stop the search, thus producing sub-optimal solutions.

The paper is structured as follows. In Section 2 we introduce the basic definitions and notations for the paper. In Section 3 the fragmentation problem is motivated and formalized as a 0–1 IP

problem. Section 4 presents a large set of experimental results based on the TPC-H benchmark. In Section 5 the related work is surveyed, while Section 6 draws the conclusions. Appendix A describes the cost function used for optimization.

2. Background

This section introduces some necessary background knowledge for the paper. In particular, Section 2.1 introduces the basics on cube schemes, Section 2.2 discusses the assumptions made on the workload for the MD, and Section 2.3 identifies the candidate views for materialization.

2.1. Cubes

Definition 1 (*Cube scheme and cube*). A *cube scheme* $\mathscr{C}$ is defined by a set of *dimensions*, $Gby(\mathscr{C})$, and a set of *measures*, $Meas(\mathscr{C})$. A *cube* instance of a cube scheme is a partial function which maps from the Cartesian product of the domains of dimensions to the Cartesian product of the domains of measures.

Typically, each dimension is associated with a set of *attributes* which describe it; these attributes are organized into a directed tree, called *hierarchy*, whose root is the dimension and whose arcs represent functional dependencies.

Definition 2 (*Multidimensional scheme*). A *multidimensional scheme* $\mathscr{D}$ is a set of cube schemes.

Example 1. The multidimensional scheme we will use as a working example derives from the TPC-H [13]; it includes two cube schemes, *LineItem* (*LI*) and *PartSupplier* (*PS*), which respectively describe the composition of the orders issued from a company and the company supplies:

$Gby(LI) = \{$Part, Supplier, Order, ShipDate, ShipMode, ReturnFlag, Status,

            CommitDate, ReceiptDate$\}$

$Meas(LI) = \{$UnitPrice, Qty, ExtPrice, Discount, DiscPrice, Charge, Tax$\}$

$Gby(PS) = \{$Part, Supplier, Date$\}$

$Meas(PS) = \{$AvailQty, SupplyCost$\}$

LI and *PS* are characterized by the hierarchies represented graphically in Fig. 2 where, for instance, Supplier → SNation, and SNation → SRegion. On relational DBMSs, cube schemes are usually implemented through a star scheme [2]. The star scheme for *LI* is:

PART(<u>Part</u>, Brand, MFGR, Type, Container, Size)
SUPPLIER(<u>Supplier</u>, SNation, SRegion)
ORDER(<u>Order</u>, ODate, OMonth, OQuarter, OYear, OWeek, Customer, CNation, CRegion,
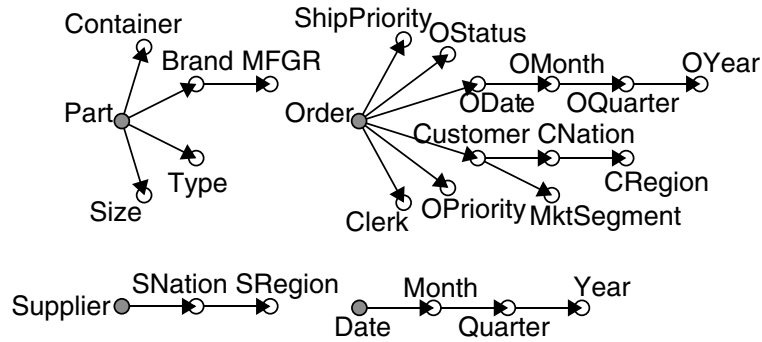   MktSegment, OPriority, OStatus, ShipPriority, Clerk)

Fig. 2. Hierarchies in the *LI* and *PS* schemes (in gray the dimensions).

LINEITEM(Part, Supplier, Order, ShipDate, ShipMode, ReturnFlag, ReceiptDate, CommitDate, Status, UnitPrice, Qty, ExtPrice, Discount, DiscPrice, Charge, Tax)

where LINEITEM is the fact table and the others are dimension tables. [1]

2.2. The workload

In principle, the workload for an MD (meant as the set of queries commonly formulated on the MD) is dynamic and unpredictable. Nevertheless, in agreement with several research papers (for instance [5,14]), we claim that a core workload can be determined a priori. In fact, on the one hand, the user typically knows in advance which kind of data analysis (s)he will carry out more often for decisional or statistical purposes; on the other, a substantial amount of queries are aimed at extracting summary data to fill standard reports. Furthermore, some commercial tools enable the workload to be monitored while the MD is operating: in this case, fragmentation could be carried out periodically considering the current workload in order to tune performances at best.

In the following we will consider the GPSJ class of queries, very common in OLAP applications. A GPSJ (generalized projection/selection/join) query is a selection over a generalized projection over a selection over a join, where the generalized projection operator is an extension of duplicate eliminating projection which captures grouping and aggregation [15]. In a GPSJ query within the OLAP context, the join relates one or more fact tables with the dimension tables; the selection may be applied to both measures (non-key attributes of the fact table) and attributes (non-key attributes of dimension tables); the generalized projection groups the tuples on a set of attributes (*grouping set*) and determines which aggregated measures are returned. Given query q, we will denote with $Gby(q)$, $Meas(q)$, $Sel(q) \in [0 \ldots 1]$, and $Freq(q) \in [0 \ldots 1]$, respectively, the grouping set of q, the measures it returns, its selectivity, and its frequency within the workload.

Given a cube scheme $\mathscr{C}$, let $\mathscr{G}_{\mathscr{C}}$ denote the set of the grouping sets of all the possible queries on $\mathscr{C}$. A query q on $\mathscr{C}$ is well-formed if $Meas(q) \subseteq Meas(\mathscr{C})$ and $Gby(q) \in \mathscr{G}_{\mathscr{C}}$.

---

[1] We will assume for simplicity that the primary key of each dimension table is a dimension. In practice, surrogate keys are typically introduced instead.

Example 2. On *LI*, the query asking for the total quantity of each medium polished part ordered from American suppliers is characterized by $Gby(q) = \{SRegion, Part\}$, $Meas(q) = \{Qty\}$ and $Sel(q) = 0.01$ (assuming that 5 supplier regions and 20 part types are present, that attribute values are uniformly distributed and that selection predicates are independent, it is $Sel(q) = \frac{1}{5} \cdot \frac{1}{20}$). Its SQL formulation on the star scheme defined in Example 1 is:

```
SELECT   S.SRegion, P.Part, SUM(LI.Qty)
FROM     LINEITEM AS LI, PART AS P, SUPPLIER AS S
WHERE    LI.Part = P.Part AND LI.Supplier = S.Supplier
AND      P.Type = 'Medium Polished' AND S.SRegion = 'USA'
GROUP BY S.SRegion, P.Part
```

A query *q* may also require measures taken from distinct cube schemes $\mathscr{C}_1, \ldots, \mathscr{C}_m$. In this case, *q* is well-formed if $Meas(q) \subseteq \bigcup_{k=1}^{m} Meas(\mathscr{C}_k)$ and $Gby(q) \in \bigcap_{k=1}^{m} \mathscr{G}_{\mathscr{C}_k}$.

Example 3. The query which compares the total available quantity and the total quantity sold for each part, characterized by $Gby(q) = \{Part\}$, $Meas(q) = \{AvailQty, Qty\}$, and $Sel(q) = 1$, is formulated in SQL as follows:

```
SELECT   P.Part, SUM(PS.AvailQty), SUM(LI.Qty)
FROM     LINEITEM AS LI, PART AS P, PARTSUPPLIER AS PS
WHERE    LI.Part = P.Part
AND      PS.Part = P.Part
GROUP BY P.Part
```

As observed in [3], the functional dependencies which relate the attributes of hierarchies in $\mathscr{C}$ induce a partial ordering, called *roll-up* ($\preceq$), on $\mathscr{G}_{\mathscr{C}}$: given two grouping sets g_i and g_j, it is $g_i \preceq g_j$ iff $g_j \rightarrow g_i$ (g_j functionally determines g_i). The roll-up ordering identifies a lattice in which the top element is $Gby(\mathscr{C})$ (the finest grouping set) and the bottom element is the empty grouping set, $\emptyset$ (the coarsest possible one). While $Gby(\mathscr{C})$ will be called the *primary grouping set* of $\mathscr{C}$, all the other grouping sets in $\mathscr{G}_{\mathscr{C}}$ will be called the *secondary grouping sets* of $\mathscr{C}$.

Example 4. Some examples of grouping sets on *LI* are

$$Gby(LI) = \{Part, Supplier, Order, ShipDate, Status, ShipMode, CommitDate,$$
$$ReturnFlag, ReceiptDate\}$$

$$g_1 = \{Part, OMonth, SNation\}$$

$$g_2 = \{Brand, Type\}$$

$$g_3 = \{OYear, SNation\}$$

$$g_4 = \emptyset$$

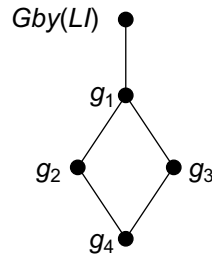The roll-up relationships between them are summarized in Fig. 3.

Fig. 3. Roll-up relationships between grouping sets on the *LI* scheme.

2.3. Candidate views

In MDs, the response to the workload is typically improved by materializing, besides the primary view storing elemental data, a set of *secondary views* which store the pre-computed results to frequent/critical queries. In the classical approaches to materialization, each view includes all and only the measures of one cube scheme; thus, in principle, each grouping set on a given cube scheme determines exactly one possible view to be materialized.

Several algorithms have been proposed to determine the set of materialized views which minimizes the workload cost by respecting a constraint posed on the total storing space. Most approaches [3,4,10] start by determining the set of views which may potentially reduce the cost for executing the workload (*candidate views*);[2] then, the views to be materialized are chosen by applying some (exact or heuristic) optimization technique to the set of candidate views. In particular, in [3] the authors supply a criterion for defining candidate views and prove that materializing a non-candidate view can never decrease the workload cost.

In the remainder we will assume that a set of candidate views has been determined for each cube scheme involved in the workload. Let $Cand(\mathscr{C})$ denote the set of grouping sets of the candidate views (*candidate grouping sets*) determined for cube scheme $\mathscr{C}$. For each cube scheme, every measure must be materialized at the primary grouping set (i.e. in its non-aggregated form); thus, $Gby(\mathscr{C}) \in Cand(\mathscr{C})$ for every $\mathscr{C}$.

3. Materialization of fragmented views

While in the classical approach to materialization each view is univocally characterized by its grouping set and by the cube scheme it is computed from, in our approach a view (which will be more properly called *fragment*) is characterized by its grouping set and by a set of measures belonging to the cube schemes in the multidimensional scheme.

---

[2] For instance, a view whose grouping set is coarser than the grouping sets of all the queries in the workload is never useful, thus it is not candidate.

Definition 3 (*Fragment*). Given multidimensional scheme $\mathscr{D}$, a fragment $\mathscr{F}$ on $\mathscr{D}$ is defined by a grouping set $Gby(\mathscr{F})$ and by a set of measures $Meas(\mathscr{F})$, where $Gby(\mathscr{F}) \in \mathscr{G}_{\mathscr{C}_k}$ for each $\mathscr{C}_k \in \mathscr{D}$ such that $Meas(\mathscr{F}) \cap Meas(\mathscr{C}_k) \neq \emptyset$.

In terms of relational implementation, fragment $\mathscr{F}$ corresponds to a fact table having the attributes in $Gby(\mathscr{F})$ as the key and the measures in $Meas(\mathscr{F})$ as non-key attributes.

Example 5. The fact table for fragment $\mathscr{F}$ with $Gby(\mathscr{F}) = \{\text{Part}\}$ and $Meas(\mathscr{F}) = \{\text{AvailQty}, \text{Qty}\}$ has scheme

LINEITEM_F(<u>Part</u>,AvailQty, Qty)

and is defined by the query in Example 3.

Definition 4 (*Fragmentation*). Let $Cand(\mathscr{C}_k)$ be the candidate grouping sets for $\mathscr{C}_k \in \mathscr{D}$. A *fragmentation* on $\mathscr{D}$ is a set $\mathscr{H}$ of fragments with the following properties:

(1) *Consistent*. A fragment always corresponds to a candidate grouping set:

$$\forall \mathscr{F}_i \in \mathscr{H}, \quad \forall m_s \in Meas(\mathscr{F}_i) \quad \exists \mathscr{C}_k \in \mathscr{D} : Gby(\mathscr{F}_i) \in Cand(\mathscr{C}_k) \wedge m_s \in Meas(\mathscr{C}_k) \tag{1}$$

(2) *Lossless*. Each measure of every cube scheme is materialized at least at its primary grouping set:

$$\forall \mathscr{C}_k \in \mathscr{D}, \quad \forall m_s \in Meas(\mathscr{C}_k) \quad \exists \mathscr{F}_i \in \mathscr{H} : Gby(\mathscr{F}_i) = Gby(\mathscr{C}_k) \wedge m_s \in Meas(\mathscr{F}_i) \tag{2}$$

(3) *Non-redundant*. No overlap between fragments is allowed:

$$\forall \mathscr{F}_i, \mathscr{F}_j \in \mathscr{H} \; (Gby(\mathscr{F}_i) = Gby(\mathscr{F}_j)) \Rightarrow (Meas(\mathscr{F}_i) \cap Meas(\mathscr{F}_j) = \emptyset) \tag{3}$$

Constraint (1) is meant to enforce the correspondence between cubes and grouping sets. The reason for constraint (2) is to ensure that every possible query on $\mathscr{D}$ can be answered on $\mathscr{H}$. Finally, the reason for constraint (3) is that a redundant fragmentation would heavily impact on query processing.

Several fragments, even defined on different grouping sets, may be necessary to solve a given query. Given a subset of fragments $\mathscr{H}' \subset \mathscr{H}$, query q can be answered on $\mathscr{H}'$ iff

$$\forall m_s \in Meas(q) \quad \exists \mathscr{F}_i \in \mathscr{H}' : m_s \in Meas(\mathscr{F}_i) \wedge Gby(q) \preceq Gby(\mathscr{F}_i) \tag{4}$$

If q necessarily requires two or more fragments (since no fragment includes all the measures in $Meas(q)$), it must be solved by first retrieving the necessary measures from the single fragments, then performing a join between the results (the join attributes are those in $Gby(q)$).

In the remainder of this section we face the problem of determining the fragmentation which minimizes the workload cost satisfying a given constraint on the global space for materialization. After emphasizing the motivations for fragmentation by proposing a quantitative example on the TPC-H benchmark, we will introduce the problem formulation and propose a mathematical programming approach to determine the optimal fragmentation.

3.1. Fragmentation vs. classical materialization: an example

In the classic approach to materialization, selecting a candidate grouping set $g \in Cand(\mathscr{C})$ during the optimization process means materializing a view with grouping set g and measures $Meas(\mathscr{C})$. As a result, each view includes measures which describe the same cube scheme but, within the workload, may be often requested separately. We argue that the system overall response to the workload may be increased by materializing fragments which include all and only the measures which typically appear together within the queries.

Let the workload on LI and PS include three queries defined as follows:

$Meas(q_1) = \{\mathsf{DiscPrice}, \mathsf{Charge}, \mathsf{Tax}\}$,

$Gby(q_1) = \{\mathsf{Brand}, \mathsf{Supplier}\}$;

$Meas(q_2) = \{\mathsf{UnitPrice}, \mathsf{Qty}, \mathsf{ExtPrice}, \mathsf{Discount}\}$,

$Gby(q_2) = \{\mathsf{Part}, \mathsf{Supplier}, \mathsf{ShipDate}\}$;

$Meas(q_3) = \{\mathsf{Qty}, \mathsf{AvailQty}\}$,

$Gby(q_3) = \{\mathsf{Part}, \mathsf{Supplier}\}$.

While q_1 and q_2 are formulated on LI, q_3 is formulated on LI and PS; all selectivities are assumed equal to 1. The candidate grouping sets for this workload are

$Cand(LI) = \{Gby(LI), Gby(q_1), Gby(q_2), Gby(q_3)\}$,

$Cand(PS) = \{Gby(PS), Gby(q_3)\}$.

Using a classical approach, the possible secondary views to materialize are $\mathscr{V}_1$, $\mathscr{V}_2$, $\mathscr{V}_3$, and $\mathscr{V}_4$ characterized by

$Meas(\mathscr{V}_1) = Meas(\mathscr{V}_2) = Meas(\mathscr{V}_3)$
$\qquad\quad = \{\mathsf{UnitPrice}, \mathsf{Qty}, \mathsf{ExtPrice}, \mathsf{Discount}, \mathsf{DiscPrice}, \mathsf{Charge}, \mathsf{Tax}\}$,

$Meas(\mathscr{V}_4) = \{\mathsf{AvailQty}, \mathsf{SupplyCost}\}$;

$Gby(\mathscr{V}_1) = Gby(q_1)$,

$Gby(\mathscr{V}_2) = Gby(q_2)$,

$Gby(\mathscr{V}_3) = Gby(\mathscr{V}_4) = Gby(q_3)$.

Consider now the five fragments $\mathscr{F}_1, \ldots, \mathscr{F}_5$ defined as follows:

$Meas(\mathscr{F}_1) = \{\mathsf{DiscPrice}, \mathsf{Charge}, \mathsf{Tax}\}$,

$Gby(\mathscr{F}_1) = Gby(q_1)$;

$Meas(\mathscr{F}_2) = \{\mathsf{UnitPrice}, \mathsf{Qty}, \mathsf{ExtPrice}, \mathsf{Discount}\}$,

$Gby(\mathscr{F}_2) = Gby(q_2)$;

$Meas(\mathscr{F}_3) = \{Qty, AvailQty\},$

$Gby(\mathscr{F}_3) = Gby(q_3);$

$Meas(\mathscr{F}_4) = \{UnitPrice, Qty, ExtPrice, Discount, AvailQty\},$

$Gby(\mathscr{F}_4) = Gby(q_2);$

$Meas(\mathscr{F}_5) = \{AvailQty\},$

$Gby(\mathscr{F}_5) = Gby(q_3).$

The costs for executing each query on each view and fragment are summarized in Table 1. The function cost used expresses the number of disk pages in which the tuples required to answer the query are contained, and is detailed in Appendix A. In Table 2 some materialization solutions are compared in terms of total disk space occupied and workload cost; primary views are not considered, assuming that they will be materialized within all solutions. In particular, materializing three subsets of fragments is compared to materializing four subsets of views.

It clearly appears how fragmentation may reduce the overall workload cost since:

- While each view includes all the measures in its cube scheme, each fragment only includes those measures which, aggregated at the given grouping set, are useful for the workload.

Table 1
Execution costs

View/fragment	q_1	q_2	q_3
$\mathscr{V}_1$	2084	–	–
$\mathscr{V}_2$	54 932	54 932	(54 932)
$\mathscr{V}_3$	6934	–	(6934)
$\mathscr{V}_4$	–	–	(2051)
$\mathscr{F}_1$	1028	–	–
$\mathscr{F}_2$	–	35 157	(35 157)
$\mathscr{F}_3$	–	–	2051
$\mathscr{F}_4$	–	38 086	38 086
$\mathscr{F}_5$	–	–	(1172)

A dash denotes that a query cannot be executed on a view/fragment, the costs in parentheses denote that not all the measures required by the query are found on the fragment.

Table 2
Comparison of materialization solutions

Mater. views/fragments	Size (MB)	Workload cost (pages)
$\mathscr{H}_1 = \{\mathscr{V}_2, \mathscr{V}_4\}$	456	166 847
$\mathscr{H}_2 = \{\mathscr{V}_1, \mathscr{V}_2, \mathscr{V}_4\}$	473	113 999
$\mathscr{H}_3 = \{\mathscr{V}_2, \mathscr{V}_3, \mathscr{V}_4\}$	511	70 851
$\mathscr{H}_4 = \{\mathscr{V}_1, \mathscr{V}_2, \mathscr{V}_3, \mathscr{V}_4\}$	528	66 001
$\mathscr{H}_5 = \{\mathscr{F}_1, \mathscr{F}_2, \mathscr{F}_3\}$	306	38 236
$\mathscr{H}_6 = \{\mathscr{F}_1, \mathscr{F}_4\}$	313	77 200
$\mathscr{H}_7 = \{\mathscr{F}_1, \mathscr{F}_2, \mathscr{F}_5\}$	299	72 514

- Materializing small fragments instead of large views leads to decreasing the query cost both directly, since smaller fact tables are to be read, and indirectly, since the storage space saved can be used to materialize extra fragments.
- Including measures from different cube schemes in the same fragment reduces the cost of queries since no join between fact tables is required.
- The extra space required by key duplication quickly decreases when the grouping set decreases; it is further reduced when junk dimensions [3] are present.
- The effectiveness of fragmentation for MDs may be higher than for operational non-redundant databases; in fact, in MDs the presence of fragments storing the same measures at different levels of aggregation makes multiple solutions possible.

3.2. Problem formulation

In this section we propose the *vertical fragmentation problem* (VFP) as a combinatorial optimization problem and discuss a possible IP formulation.

Recent IP results, both on exact [16] and heuristic [17,18] techniques, have in fact reached a point where even complex real-world problems can be solved in acceptable CPU time. It is thus worthwhile an investigation of efficient IP formulations, as these provide a key to directly access robust and effective solution techniques.

We begin by observing that, while in principle VFP should be solved for the whole set of cube schemes in the multidimensional scheme, in practice it may be convenient to mix measures belonging to two different cube schemes in the same fragment only if the workload includes at least one query that requires measures from both. In most cases, this allows to dramatically cut the complexity by adopting a simple divide-and-conquer approach and applying fragmentation separately on subsets of cube schemes based on partial workloads. In the following, $\mathscr{D}$ will stand for one of these subsets.

Given a workload, the measures of each cube scheme in $\mathscr{D}$ can be partitioned into subsets (*minterms*) such that all the measures in a minterm are requested together by at least one query and do not appear separately in any other query. We call *terms* the sets of measures, even of different cube schemes, recursively defined as follows:

(1) a minterm is a term;
(2) the union of two terms including measures required from the same query is a term.

For instance, given the workload on *LI* and *PS* proposed in Section 3.1, the minterms are:

$$mt_1 = \{\mathsf{DiscPrice}, \mathsf{Charge}, \mathsf{Tax}\},$$

$$mt_2 = \{\mathsf{UnitPrice}, \mathsf{ExtPrice}, \mathsf{Discount}\},$$

---

[3] In [2], the term *junk dimensions* is used to denote the case in which two or more hierarchies are represented jointly within the same dimension table.

$mt_3 = \{\text{AvailQty}\},$

$mt_4 = \{\text{Qty}\}.$

The corresponding set of terms is:

$$T = \{mt_1, mt_2, mt_3, mt_4, mt_2 \cup mt_4, mt_3 \cup mt_4, Meas(LI), Meas(PS)\}$$

Given the multidimensional scheme $\mathscr{D}$ and workload Q, let $G = \bigcup_{\mathscr{C} \in \mathscr{D}} Cand(\mathscr{C})$ be the set of all candidate grouping sets and T the set of all terms. In the following we will show how VFP can be modeled by expressing linear IP constraints over two sets of 0–1 binary variables, $\{x_{ijk}\}$ and $\{y_{jk}\}$, whose indexes correspond to the indexes of the queries $q_i \in Q$, of the grouping sets $g_j \in G$ and of the terms $t_k \in T$, respectively. Setting $y_{jk} = 1$ means stating that a fragment $\mathscr{F}_{jk}$ with grouping set $Gby(\mathscr{F}_{jk}) = g_j$ and measures $Meas(\mathscr{F}_{jk}) = t_k$ will be created. Setting $x_{ijk} = 1$ means stating that, when answering q_i, fragment $\mathscr{F}_{jk}$ will be accessed. Thus, a solution at the same time denotes a fragmentation,

$$\mathscr{H} = \{\mathscr{F}_{jk} : y_{jk} = 1\} \tag{5}$$

and specifies on which fragment(s) each query in Q is assumed to be executed. Fig. 4 shows the solution which represents fragmentation $\mathscr{H}_5$ proposed in the example in Section 3.1, featuring 3 fragments. The solution also denotes that, for instance, q_1 is executed on $\mathscr{F}_1$.
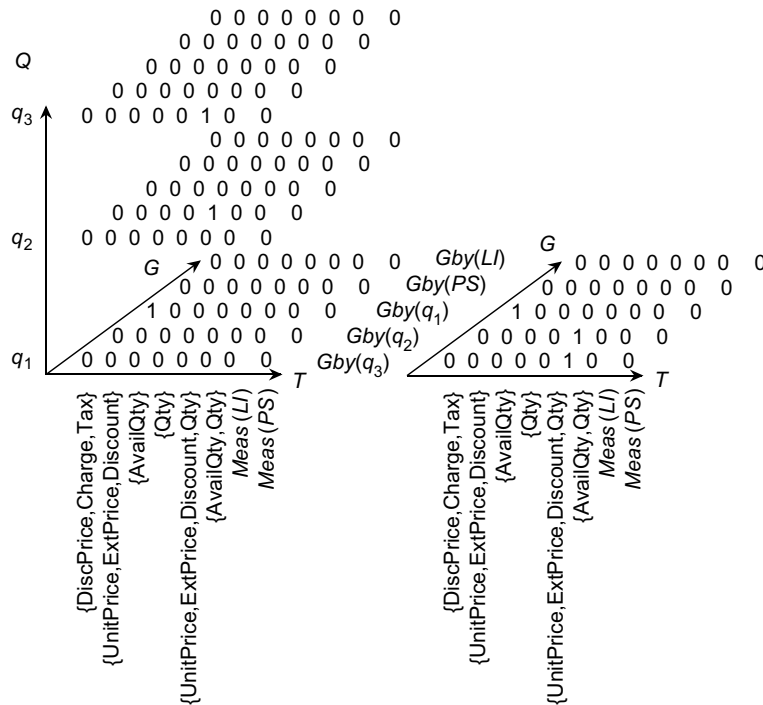

Fig. 4. Sets of the x_{ijk} (left) and y_{jk} (right) for fragmentation $\mathscr{H}_5$ in Section 3.1.

Table 3
Notation employed for index sets in the IP formulation of VFP

$\mathcal{Q}$	Index set of the queries in Q
$\mathcal{G}$	Index set of the grouping sets in G
$\mathcal{T}$	Index set of the terms in T
$\mathcal{M}$	Index set of the measures in $\bigcup_{\mathscr{C} \in \mathscr{D}} Meas(\mathscr{C})$
$\mathcal{G}_i \subseteq \mathcal{G}$	Index set of the useful grouping sets for query $i \in \mathcal{Q}$
$\mathcal{M}_i \subseteq \mathcal{M}$	Index set of the measures in $Meas(q_i)$
$\mathcal{T}_j \subseteq \mathcal{T}$	Index set of the terms at grouping set $j \in \mathcal{G}$
$_Q\mathcal{T}_{ij} \subseteq \mathcal{T}$	Index set of the useful terms for query $i \in \mathcal{Q}$ at grouping set $j \in \mathcal{G}_i$
$_M\mathcal{T}_{js} \subseteq \mathcal{T}$	Index set of the useful terms for measure $s \in \mathcal{M}$ at grouping set $j \in \mathcal{G}$

Definition 5 (*VFP*). Given a multidimensional scheme $\mathscr{D}$, a workload Q on $\mathscr{D}$, a set of candidate grouping sets $Cand(\mathscr{C})$ for each cube scheme $\mathscr{C} \in \mathscr{D}$, and a constraint B on disk space, VFP consists in determining, among the fragmentations of $\mathscr{D}$ which require no more than B disk space, the one which minimizes the cost of workload Q.

Using the notation for index sets summarized in Table 3, the IP formulation of VFP is as follows:

$$Cost(VFP) = \text{Min} \sum_{i \in \mathcal{Q}} \sum_{j \in \mathcal{G}_i} \sum_{k \in _Q\mathcal{T}_{ij}} c_{ijk} x_{ijk} \tag{6}$$

$$\text{s.t.} \sum_{j \in \mathcal{G}_i} \sum_{k \in _M\mathcal{T}_{js}} x_{ijk} \geqslant 1 \quad i \in \mathcal{Q}, \ s \in \mathcal{M}_i \tag{7}$$

$$\sum_{k \in _M\mathcal{T}_{js}} y_{jk} \leqslant 1 \quad j \in \mathcal{G}, \ s \in \mathcal{M} \tag{8}$$

$$x_{ijk} \leqslant y_{jk} \quad i \in \mathcal{Q}, \ j \in \mathcal{G}_i, \ k \in _Q\mathcal{T}_{ij} \tag{9}$$

$$\sum_{j \in \mathcal{G}} \sum_{k \in \mathcal{T}_j} b_{jk} y_{jk} \leqslant B \tag{10}$$

$$x_{ijk} \in \{0,1\} \quad i \in \mathcal{Q}, \ j \in \mathcal{G}_i, \ k \in _Q\mathcal{T}_{ij} \tag{11}$$

$$y_{jk} \in \{0,1\} \quad j \in \mathcal{G}, \ k \in \mathcal{T}_j \tag{12}$$

where:
- *Cost*(*VFP*) denotes the *optimal cost* of VFP, i.e. the cost of Q on the optimal fragmentation;
- $x_{ijk} = 1$ iff, when answering query i, the measures in term k are read at grouping set j;
- $y_{jk} = 1$ iff $x_{ijk} = 1$ for at least one query i;
- c_{ijk} is the cost for (partially) answering query i on fragment $\mathscr{F}_{jk}$ defined by g_j and t_k, i.e. for reading the relevant term k on grouping set j (estimated for instance as in formula (A.2) in Appendix A);
- b_{jk} is the disk space required to store fragment $\mathscr{F}_{jk}$ (calculated as in formula (A.1) in Appendix A).
- Given $i \in \mathcal{Q}$, $\mathcal{G}_i = \{j : Gby(q_i) \preceq g_j \wedge (\exists \mathscr{C} \in \mathscr{D} : g_j \in Cand(\mathscr{C}) \wedge Meas(q_i) \cap Meas(\mathscr{C}) \neq \emptyset)\}$ is the index set of the grouping sets which might be used to solve q_i.

- Given $j \in \mathcal{G}$ and $s \in \mathcal{M}$, $_M\mathcal{T}_{js} = \{k : m_s \in t_k \wedge (\exists \mathcal{C} \in \mathcal{D} : g_j \in C$ and $(\mathcal{C}) \wedge m_s \in Meas(\mathcal{C}))\}$ is the index set of the terms which might be used to retrieve m_s at grouping set g_j.
- Given $i \in \mathcal{Q}$ and $j \in \mathcal{G}_i$, $_Q\mathcal{T}_{ij} = \bigcup_{s \in \mathcal{M}_i} {}_M\mathcal{T}_{js}$ is the index set of the terms which might be used to solve q_i at grouping set g_j.
- Given $j \in \mathcal{G}$, the index set of the terms at grouping set g_j is $\mathcal{T}_j = \bigcup_{i \in \mathcal{Q}} {}_Q\mathcal{T}_{ij} = \bigcup_{s \in \mathcal{M}} {}_M\mathcal{T}_{js}$.

The explanation of the elements of formulation VFP is as follows. The objective function (6) states that the solution must minimize the sum of the costs for answering all queries in Q on each fragment required (see Appendix A). Inequalities (7) impose that each measure required by a query is read on at least one grouping set, i.e. that the solution satisfies the workload. Inequalities (8) require that, for each grouping set, a measure belongs to at most one term, i.e. that the fragmentation is non-redundant as stated in property (3) of Definition 4. Inequalities (9) link the x and y variables and inequality (10) is the memory knapsack constraint, stating that the disk space required by fragmentation must be less than B. Finally, (11) and (12) are the integrality constraints. As to the other fragmentation properties, consistency (1) is implicitly expressed by formulating the IP constraints on the restricted index sets $\mathcal{G}_i$, $_Q\mathcal{T}_{ij}$, and $_M\mathcal{T}_{js}$ instead of their supersets $\mathcal{G}$ and $\mathcal{T}$. Lossless fragmentation (2) is enforced by inserting in the workload, for each cube scheme $\mathcal{C}$, a dummy query $q_\mathcal{C}$ such that $Meas(q_\mathcal{C}) = Meas(\mathcal{C})$, $Gby(q_\mathcal{C}) = Gby(\mathcal{C})$, and $Freq(q_\mathcal{C}) = 0$: this trick allows for making fragmentation lossless without altering the workload cost.

Notice the structure of VFP. There are essentially three blocks of constraints: inequalities (7), (8) and (10). Inequalities (7), the query satisfaction constraints, are actually *set covering* constraints imposing that exactly one x_{ijk} variable enters the solution for each i and for each measure in $\mathcal{M}_i$: problem costs are attached only to variables x_{ijk}, thus this set covering part directly affects the objective cost. Inequality (10), as mentioned, is a *knapsack* constraint, while inequalities (8) are needed to impose measure-term surjectivity in the form of *set packing* constraints. Problem VFP is NP-hard, since when no constraint is imposed on memory (i.e. B is very big) and only one query $i \in \mathcal{Q}$ is defined for each $j \in \mathcal{G}_i$ and $k \in {}_Q\mathcal{T}_{ij}$, it becomes a standard set covering problem, which is NP-hard. When memory is tight, VFP is further complicated by the interplay of set covering, set packing and knapsack.

With reference to the example introduced in Section 3.1 and solved in Fig. 4, the different constraints impose the following. Constraints (7) essentially force each G–T bit plane in the left of the figure to have at least one "1" for each measure required by the query associated to it, but the more they are the more the solution costs. Constraints (8) require that for each row on the right-hand side bit plane there is at most one "1" corresponding to terms with common measures. Constraints (9) impose that whenever there is a "1" in any bit plane on the left of the figure, there must be a "1" also in the corresponding G–T position in the array on the right, whose values are then used to check the memory constraint (10), not be shown in the figure, that asks, having computed the b_{ij} load associated with each y_{ij} element of the G–T bit plane, to choose a subset of the y_{ij} elements whose total associated load does not exceed a given parameter B.

By a linear relaxation of integrality constraints, that is by substituting constraints (11) and (12) with:

$$x_{ijk} \leqslant 1 \quad i \in \mathcal{Q}, j \in \mathcal{G}_i, k \in {}_Q\mathcal{T}_{ij} \tag{13}$$

$$y_{jk} \leqslant 1 \quad j \in \mathcal{G}, k \in \mathcal{T}_j \tag{14}$$

we get problem LVFP: a linear problem that can be solved in polynomial time and whose optimal solution value $Cost(LVFP)$ constitutes a lower bound to $Cost(VFP)$, the optimal solution cost of problem VFP.

The formulation proposed closely matches the problem description and at the same time shows common features of good mathematical formulations [19]: it uses as few variables as possible, it needs a polynomial number of constraints, it allows for efficient decompositions, and it allows for a tight linear relaxation. Nevertheless, different mathematical formulations are obviously possible. For example, formulation VFP can be equivalently translated into formulation VFP′, where constraints (9) are surrogated over the queries: this does not affect the integer optimum, however it could affect the linear relaxation, as a much greater variable fragmentation could result. Yet another formulation was used in [11] where, instead of constraints (7), equality constraints were used: this implies that a subproblem to solve is a set partitioning problem, one of the nastiest problems one could happen to meet, and in fact the computational results in this paper substantially improve over those reported in [11].

4. Experimental results

An extensive set of tests have been carried out in order to verify the effectiveness of fragmentation. Tests are aimed at comparing our approach to the classic solution to view materialization and at emphasizing the specific characteristics of the fragmented solutions. Tests are based on the well-known TPC-H benchmark [13], sized at 1 GB and loaded on Red Brick Decision Server 6.0. The operational scheme proposed within TPC-H has been used to obtain the two cube schemes, *LI* and *PS*, whose simplified versions have been used as working example in the previous sections. In our implementation, *LI* consists of 7 hierarchies with 40 attributes and 10 measures, *PS* of 3 hierarchies with 20 attributes and 3 measures.

The tests are organized according to three different workloads with increasing size. Q_1 (20 queries) tightly models the TPC-H benchmark including all the queries that can be formulated as GPSJ expressions, as well as the GPSJ subqueries that are contained in non-GPSJ ones. Q_2 and Q_3 (30 and 40 queries, respectively) extend TPC-H by adding new queries with different grouping sets and whose measures are chosen consistently with the application logic defined in the benchmark. For each workload, the bulk of the queries works on *LI*, while the percentage of queries requiring both schemes ranges between 15% and 20%.

Maintaining a realistic clustering of measures in the test workloads is important to avoid to excessively emphasize the effect of fragmentation. On the other hand, measures are naturally clustered by the level of correlation of the information they carry. The level of *affinity* of a couple of measures, m_i and m_j, within workload Q can be evaluated as the percentage of queries that require both m_i and m_j. For all three workloads we used, the average affinity is about 0.2.

In the following subsections, the workload execution cost is expressed in terms of disk pages. It is remarkable that all the data presented in the tests were directly measured on the DBMS rather than obtained from simulations, thus avoiding that the simplifications induced by the cost function proposed in Appendix A invalidate our results. As to indexing of data, we adopted a standard solution which provides a B$^+$-Tree index on each primary key of both fact and dimension tables.

The algorithms for view materialization and fragmentation are implemented within WAND [20], the data warehouse design tool we developed. In selecting the classic approach to materialization to be compared with fragmentation, we observed that: (1) all approaches to materialization are characterized by an objective function to be optimized and an optimization algorithm; (2) any objective function could be adopted in principle, but the comparison is sound only if the same function used for fragmentation is adopted; (3) as to the optimization algorithm, all approaches to materialization we are aware of are based on heuristic algorithms which deliver suboptimal solutions [21]. Thus, adopting one or the other of the materialization algorithms proposed in the literature would not alter significantly the comparison results: the one we implemented is described in [3] and selects views using a tabu search heuristics; the objective function is the one described in Appendix A, which estimates the execution cost of the workload.

4.1. Benefit

The aim of this set of tests is to analyze to which extent fragmentation reduces the workload cost and which factors determine the reduction. Fig. 5 compares the execution costs for the fragmented and unfragmented solution of Q_1, Q_2, and Q_3. For each workload, the space constraint ranges from 700 to 2200 MB; since the primary views globally require 640 MB, the space reserved for secondary views varies from 60 to 1560 MB.

The cost reduction, that ranges from 18% to 73% and is 38% in the average, is due to two interrelated factors: on the one side, each query requires less disk pages to be answered since only relevant measures are stored in fragments; on the other, the saved space can be used to materialize more fragments which further enhance performances. It should be noted that the previously mentioned factors participate differently in reducing the cost, according to the available space. When only a limited space is available for materialization, most of the saving comes from the reduced size of fragments while, when more space is available, the saving is mainly due to the execution of queries on fragments that do not fit the disk space constraint in the classic solution. The rationale for this behavior is that, the more the materialized views, the more the disk space saved by dropping unuseful measures, the more the extra fragments materialized; this is confirmed by Fig. 6 which compares the number of fragments and the number of views for different space constraints, showing that the former increases more quickly than the latter.

4.2. Processing

In MDs, when views have been materialized at different aggregation levels, query processing typically relies on an *aggregate navigator* capable of selecting, among the different views on which each query can be solved, the cheapest one. In the absence of fragmentation, the smallest view is generally chosen to this end.

When fragmentation has been carried out, the problem of choosing the best fragments for answering a query is apparently harder. For instance, let A, B, C, D be disjoint sets of measures, and let query q require all measures in $A \cup B$ (for brevity, AB). Let three fragments be materialized:
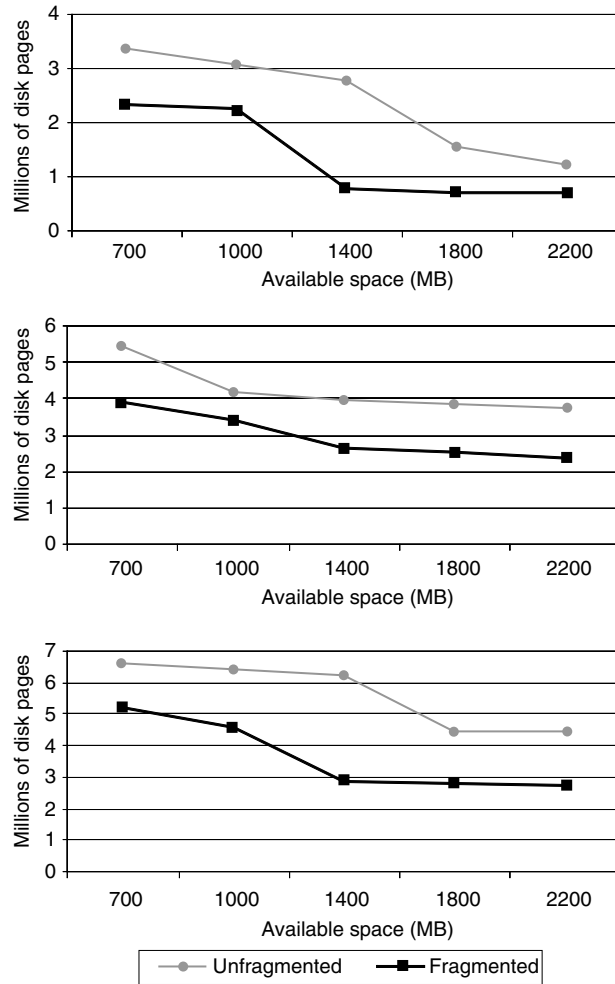
$$Meas(\mathcal{F}_1) = AC,$$

Fig. 5. Fragmented and unfragmented workload costs for Q_1 (top), Q_2 (middle), and Q_3 (bottom) varying the space constraints.

$$Gby(\mathscr{F}_1) = Gby(q);$$

$$Meas(\mathscr{F}_2) = BD,$$

$$Gby(\mathscr{F}_2) = Gby(q);$$

$$Meas(\mathscr{F}_3) = AB,$$

$$Gby(\mathscr{F}_3) = g,$$

where $Gby(q) \preceq g$. In this situation, the optimizer should choose whether to answer q on fragment $\mathscr{F}_3$ only, or to answer it by accessing both $\mathscr{F}_1$ and $\mathscr{F}_2$: in the first case, the number of pages to be
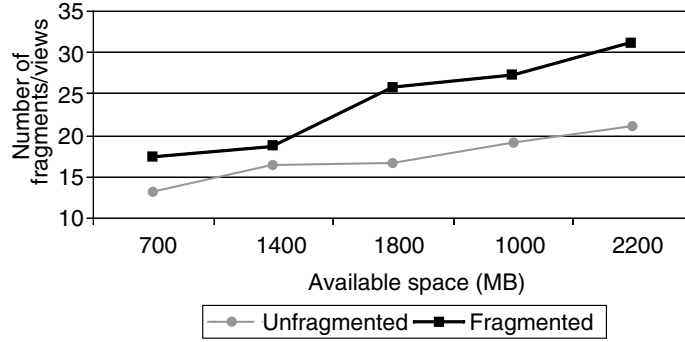
Fig. 6. Number of fragments and views materialized for different disk space constraints (data are averaged on the three workloads).

read could be high due to the high cardinality of the fragment; in the second, it could be high due to the presence of additional (non-required) measures.

The results reported in the previous subsection were computed assuming that an "ideal" aggregate navigator was available, i.e. that each query was answered on the fragment(s) yielding the minimum execution cost. This implies that the navigator should solve, for each query formulated, a complex optimization problem, which would lead to a significant overhead on query processing.

In practice, in order to limit the complexity of query processing, a reasonable heuristics must be devised. Considering the peculiar characteristics of MDs, a good polynomial heuristics consist in retrieving each measure required from the coarsest available fragment. With reference to the example above, such heuristics would answer q on $\mathscr{F}_1$ and $\mathscr{F}_2$. In the following, we give a quantitative justification to assess the validity of this approach.

Considering again the general example above, we can roughly estimate the execution costs for the two solutions as the total amounts of data to be read. Let K_i denote the primary key of fragment $\mathscr{F}_i$:

$$Cost_q(\mathscr{F}_1, \mathscr{F}_2) = Card(\mathscr{F}_1) \times (len(AC) + len(BD) + 2len(K_1))$$

$$Cost_q(\mathscr{F}_3) = Card(\mathscr{F}_3) \times (len(AB) + len(K_3))$$

where function $Card(\mathscr{F}_i)$ returns the cardinality of a fragment and function $len()$ returns the length in bytes of a set of attributes. [4] The first solution, i.e. the one returned by the heuristics, is convenient when $Cost_q(\mathscr{F}_1, \mathscr{F}_2) < Cost_q(\mathscr{F}_3)$, i.e.:

$$\frac{Card(\mathscr{F}_3)}{Card(\mathscr{F}_1)} > \frac{len(ABCD) + 2len(K_1)}{len(AB) + len(K_3)}$$

The algorithms for choosing candidate views tend to avoid materializing views with similar grouping sets [3]. Hence, it commonly is $Card(\mathscr{F}_1) \ll Card(\mathscr{F}_3)$ (at least one order of magnitude)

---

[4] Note that $Card(\mathscr{F}_1) = Card(\mathscr{F}_2)$ and $len(K_1) = len(K_2)$ since the two fragments share the same grouping set.

and the inequality will most likely be satisfied since (1) it is almost impossible that $len(CD) \gg len(AB)$; and (2) it is $len(K_3) \geqslant len(K_1)$.

The validity of this approach is confirmed by experimental results showing that, with reference to the workloads introduced in Section 4, adopting the heuristics leads to a neglectable increase in the execution costs as compared to those reported in Fig. 5.

4.3. Optimality

Solving the fragmentation problem on the full set of candidate views becomes unfeasible for large workloads; the solution we adopted, proposed in [3], preliminarily reduces the cardinality of each set by repeatedly dropping the candidate views whose grouping set is very close to the one of another candidate view. The reason for this heuristic criterion, that is well-known and accepted by data warehouse designers [2], relies on the observation that it is useless to materialize fragments/views on very similar grouping sets. When comparing fragmented solutions to unfragmented ones, the same starting set of candidate views is always used. Table 4 shows how the preliminar reduction of the candidate view sets affects the solution optimality. All test has been carried out on Q_1 and show that this heuristic criterion actually degrades the solution only in the classic approach when dramatic reductions are operated. It is remarkable that, even when significant reductions are applied, the extra flexibility induced by fragmentation limits the shift from optimality.

4.4. Impact on design

In principle, within the optimal materialization solution, when no constraints on space are posed, each query in the workload should be answered on a view with the same grouping set required by the query. In the classic approach, this is achieved by materializing one view for each grouping set required by at least one query; in the fragmented approach, materializing only the subset of relevant measures will suffice. Consequently, as confirmed at first glance by analyzing the graphs, fragmented curves flatten for smaller disk space than unfragmented ones. When the space constraint is not fixed a priori and is left to the designer's choice, the best trade-off between performance and space should be determined by a threshold on the extra-space benefit (i.e. such that the cost decreases for each MB further devoted to materialization); in the fragmented approach, such threshold will be crossed for lower disk spaces.

Table 4
Shift from optimality induced by the heuristic reduction of the set of candidate views on the materialization solution for Q_1 with 1.4 GB available

Number of views	Cost		Shift from optimality	
	Unfragmented	Fragmented	Unfragmented (%)	Fragmented (%)
100	3 286 453	538 372	256	6
300	1 284 958	520 647	0	2
500	1 284 958	510 230	0	0
743	1 284 958	510 230	0	0

4.5. Robustness

Although it is very likely that the core workload is available to the designer a priori, it is not possible to know in advance all the queries that will be submitted to the warehouse. Extemporary queries formulated by the users during OLAP sessions may decrease the global saving induced by the vertical fragmentation solution which is tailored on a specific workload. The effects of extemporary queries have been experimentally evaluated by comparing the performances of the unfragmented and fragmented solutions on workloads containing also queries not included in the reference workloads used for optimization. In particular, in Table 5 we compare the costs for executing $Q_2(Q_3)$ on the (fragmented and unfragmented) materialization solutions optimized for $Q_2(Q_3)$ and for Q_1, respectively. It is apparent that, as we might expect, the workload cost rises quickly for both the fragmented and unfragmented solutions as soon as extemporary queries are considered. When increasing their percentage, the performance for the fragmented solution decays a little more quickly than that of the unfragmented solution: in our tests, the two costs are comparable when 40% of the queries are extemporary. The reason for this behavior is that, though the extra-measures included in each view can be potentially useful to solve extemporary queries, its grouping set is often too coarse. Furthermore, the probability that a view can be used to answer a query increases for views with finer grouping sets which, unfortunately, determine lower cost savings.

4.6. Storage space

One of the main objections often moved to vertical fragmentation is that it causes a waste of disk space due to replication of keys. We wish to emphasize that our approach is based on reducing the size of fact tables by decreasing the number of measures included, while key replication occurs as a side effect only when multiple fragments are materialized on the same cube scheme and on the same grouping set. This is confirmed by the results reported in this section, where the incidence of key replication on fragmentation effectiveness is evaluated.

When the space constraint ranges from 700 to 2200 MB, the average number of surrogate keys composing the primary key of fragments varies between 2.5 and 3. Some dimensions are completely grouped, which proves that fragments are, in the average, much coarser than primary views. In fact:

- OLAP queries are intrinsically aggregated in order to allow summary information to be easily analyzed by a human user.

Table 5
Cost of a workload when executed on the materialization solution optimized for itself and for Q_1 (with space constraint 1 GB)

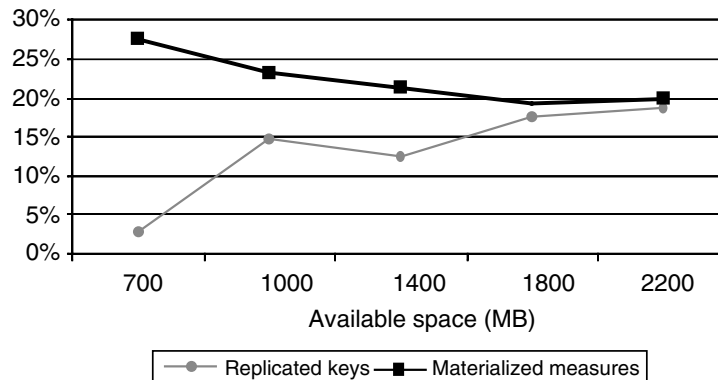| Materialization solution	Q_2		Q_3	
	Optimized for Q_2	Optimized for Q_1	Optimized for Q_3	Optimized for Q_1
Unfragmented	4138	12 989	6424	49 537
Fragmented	3411	11 204	4565	52 772

Fig. 7. Percentage of replicated keys and percentage of measures (data are averaged on the three workloads).

- The cost reduction induced by materialization is due to the reduced number of tuples that are read when answering a query on a view. On the other hand, due to the sparsity of primary views, only coarse grouping sets actually produce a significant reduction of the cardinalities of secondary views. Thus, most views are materialized on very coarse grouping sets.

The shortness of surrogate keys reduces the effects of redundancy since the space devoted to them is percentually lower for coarse fragments. As already stated, the effects of redundancy are further smoothed since our approach is inclined to materialize one fragment on each grouping set rather than more fragments on the same one. This behavior is confirmed by Figs. 6 and 7 showing that the increase in the number of fragments is much faster than the increase in the number of replicated keys when rising the space constraint. The percentage of replicated keys (calculated as the fraction of the total number of attributes in the fact tables) is lower for tight constraints and never exceeds 19%.

Fig. 7 also shows the average percentage ratio between the number of measures materialized in the fragmented and in the unfragmented solutions, which decreases when more space is available since fragments become more specialized (i.e. few queries are answered on each fragment).

4.7. Computation

We implemented an exact methodology for solving instances of VFP. The computational testing was performed using a commercial IP solver (CPLEX 6.6 by Ilog Inc.) to produce first an optimal LVFP solution, thus a lower bound, and then the optimal integer solution. All tests were run on an Intel platform with 512 MB of RAM, running at 933 MHz under Windows Me. We first generated some simple instances to fine-tune the CPLEX parameters and then solved the instances described above.

We directly applied an IP solver, i.e., a general program which is able to solve any problem expressed as a (Mixed) IP problem, instead of writing a more tailored code because formulation VFP is sufficiently well structured to permit to exploit the effectiveness of the solution procedures implemented in CPLEX. It is surely possible to further improve the results presented here by a

more tailored code, but the huge design and implementation work involved in obtaining a better code than the current one would probably not be repaid by the expectedly small improvement which could be achieved.

A full account of the computational testing can be found in Table 6. The columns show the following data:

- Probl: problem identifier;
- *n*: number of queries in the workload;
- *B*: size of memory constraint;
- *Cost(LVFP)*: optimal linear relaxation solution cost;
- *err*: percentage distance from IP optimality of *Cost(LVFP)*;
- *t(LVFP)*: CPU time to obtain *Cost(LVFP)*;
- *Cost(VFP)*: optimal integer solution cost;
- *t(VFP)*: CPU time to obtain *Cost(VFP)*;

Problem instances Qa to Qe are used to tune the final algorithm and can all be easily solved by CPLEX. The other instances are those described in the previous sections. Besides the increase in their dimension, it is apparent from column *t(VFP)* how they are structurally more complex than the training ones. As expected, the tighter the memory constraints, the harder the instance. This can be seen both from the higher error of the bound w.r.t. the optimal IP cost and from the time needed to get the optimal IP solution. Instances Q2S7 and Q3S7 are the hardest ones, they could be solved with the available RAM, but the CPU time needed is huge. This increased difficulty arises from the interplay of all problem constraints, whereas with loose memory requirements the active constraints are essentially the set covering ones.

5. Related work

The problem of determining the optimal partitioning given a workload has been widely investigated within the context of centralized as well as distributed database systems, considering non-redundant allocation of fragments (for instance, see [8,22,23]); unfortunately, the results reported in the literature cannot be applied here since the redundancy introduced by materializing views at different aggregation levels binds the partitioning problem to that of deciding on which view(s) each query should be executed. Thus, ad-hoc approaches must be devised for MDs.

While horizontal partitioning has been widely investigated and is currently adopted by designers [24], to the best of our knowledge only a few approaches to vertical fragmentation in MDs have been devised. In [7], fragmentation is extended to both measures and non-key dimensions (i.e. dimensions functionally determined by other dimensions); on the other hand, no algorithm for determining the optimal fragmentation is proposed. In [9], views are partitioned vertically in order to build dataindexes to enhance performance in parallel implementations of MDs; still, no suggestion is given on how to determine the optimal partitioning. In [10], materialization and fragmentation are seen as separate steps; the views are determined by the materialization algorithm and *then* fragmented, thus compromising the process overall optimality.

Table 6
Computational test summary results

Probl.	n	B	Cost(LVFP)	err	t(LVFP)	Cost(VFP)	t(VFP)
Qa	3	1400	50 475	0.00	0.05	50 475	0.00
Qb	5	1400	281 816	2.13	0.22	287 957	0.00
Qc	10	1400	65 190	0.00	0.66	65 190	0.00
Qd	10	1400	33 976	0.00	0.06	33 976	0.00
Qe	10	1400	116 463	0.00	0.11	116 463	0.00
Q1S7	20	700	322 535	13.77	63.65	374 047	847.94
Q1S10	20	1000	152 816	0.70	904.95	153 894	5308.99
Q1S14	20	1400	98 046	1.16	306.48	99 201	3327.66
Q1S18	20	1800	96 270	0.00	342.41	96 270	52.73
Q1S22	20	2200	96 270	0.00	370.59	96 270	31.30
Q2S7	30	700	564 786	11.60	346.97	638 882	24563.01
Q2S10	30	1000	353 786	0.47	1343.03	355 456	4738.76
Q2S14	30	1400	274 042	2.14	645.87	280 029	6880.08
Q2S18	30	1800	248 205	1.94	513.66	253 117	1677.59
Q2S22	30	2200	241 330	0.00	607.91	241 330	162.80
Q3S7	40	700	718 601	10.14	1824.24	799 666	76703.28
Q3S10	40	1000	464 241	0.06	653.83	464 535	8539.98
Q3S14	40	1400	376 146	1.78	551.29	382 947	3780.41
Q3S18	40	1800	336 431	1.37	680.53	341 090	4362.62
Q3S22	40	2200	316 795	0.11	668.17	317 155	2561.45

At the physical level, vertical fragmentation is partially implemented by *projection indexes* that store a single column of a relational table [25]; the correspondence between the rows of the index and the rows in the main table is transparently maintained by the DBMS. The rationale for this kind of index, which is particularly useful for selection and aggregation queries, is that scanning a smaller structure is faster than scanning the whole table. Though projection indexes are usually considered as auxiliary structures to improve retrieval efficiency, the SYBASE IQ commercial DBMS [26,27] pushes their use to the limit by *replacing* the data themselves, i.e. by storing data by columns instead of rows. Thus, in practice, SYBASE implements an extreme fragmentation in which each fragment includes a single measure. In this case, since the correspondence between values belonging to the same row is maintained by row identifiers, fragmentation does not require the replication of the key attributes.

Currently, to the best of our knowledge, no DBMS allows to include multiple columns into one projection index, thus our fragmentation technique cannot be applied at the physical level. However, given the competitive performance of SYBASE, it is easy to predict a relevant utility for a workload-based fragmentation if multiattribute projection indexes will be implemented. In this case, the optimal fragmentation could be determined using our approach by considering, in the function estimating the access costs, that keys are not replicated.

6. Conclusions

In this paper we have proposed a technique for materializing views in vertical fragments, aimed to tightly fit the reference workload. In the context of MDs, due to both the redundancy of data guaranteed by materialization and to the nature of queries, fragmentation is more complex than for traditional relational databases on the one hand, on the other it may yield more significant benefits.

The experimental tests suggest the effectiveness of our approach in reducing the workload cost as compared to the classical approach to view materialization. The robustness of fragmentation has been evaluated even with reference to extemporary queries. The tests also allowed to confute the main drawback typically ascribed to fragmentation, i.e. the replication of keys, showing that its impact on the storage requirements is not relevant. Besides, key replication could be completely avoided when applying fragmentation at the physical level, implementing it for instance by means of multiattribute projection indexes.

Appendix A. The cost function

Among all the feasible solutions to the fragmentation problem, we are interested in the one which minimizes the cost for executing the workload. We believe it is convenient to keep logical design independent on access plans in order to both provide a more general solution and reduce complexity. Thus, the cost function we propose intentionally abstracts from any assumptions on the access paths, being based on the number of disk pages in which the tuples of interest for a given query are stored.

Let q_i be a query requiring at least one measure in fragment $\mathscr{F}_{jk}$ defined by $Gby(\mathscr{F}_{jk}) = g_j$ and $Meas(\mathscr{F}_{jk}) = t_k$, and $Freq(q_i)$ be its frequency. The number of tuples of $\mathscr{F}_{jk}$ which must be accessed in order to answer q_i is $Sel(q_i) \cdot Card(\mathscr{F}_{jk})$, where $Card(\mathscr{F}_{jk})$ is the cardinality of $\mathscr{F}_{jk}$ estimated for instance as in [28]. The total number of pages in which $\mathscr{F}_{jk}$ is contained is

$$b_{jk} = \left\lceil \frac{Card(\mathscr{F}_{jk})}{\beta_{jk}} \right\rceil \tag{A.1}$$

where β_{jk} is the number of tuples per disk page for $\mathscr{F}_{jk}$. The expected, frequency-weighted number of pages in which the tuples of $\mathscr{F}_{jk}$ necessary for q_i are stored, c_{ijk}, can be estimated with the Cardenas formula Φ [29] as follows:

$$c_{ijk} = Freq(q_i) \cdot \Phi(Sel(q_i) \cdot Card(\mathscr{F}_{jk}), b_{jk}) \tag{A.2}$$

With reference to the index set notation summarized in Table 3, the cost of executing query i against the fragmentation denoted by $X = \{x_{ijk}\}$ is then estimated as the total number of disk pages which must be accessed in order to retrieve all the measures in q_i:

$$Cost_i(X) = \sum_{j \in \mathscr{G}_i} \sum_{k \in {}_Q\mathscr{T}_{ij}} c_{ijk}x_{ijk} \tag{A.3}$$

where $x_{ijk} = 1$ if, when answering q_i, the measures in $Meas(q_i) \cap t_k$ are read from $\mathscr{F}_{jk}$, $x_{ijk} = 0$ otherwise. Finally, the overall cost for workload Q is:

$$Cost(X) = \sum_{i \in \mathscr{Q}} Cost_i(X) = \sum_{i \in \mathscr{Q}} \sum_{j \in \mathscr{G}_i} \sum_{k \in {}_Q\mathscr{T}_{ij}} c_{ijk}x_{ijk} \tag{A.4}$$

This is the cost function minimized in (6) within the IP formulation of VFP. Though the actual number of pages read when executing the workload may be higher depending on the access path followed, we believe that this function represents a good trade-off between generality and accuracy.

References

[1] M. Gyssens, L.S. Lakshmanan, A foundation for multi-dimensional databases, in: Proc. 23rd VLDB, Athens, Greece, 1997, pp. 106–115.
[2] R. Kimball, The Data Warehouse Toolkit, John Wiley & Sons, 1996.
[3] E. Baralis, S. Paraboschi, E. Teniente, Materialized view selection in multidimensional database, in: Proc. 23rd VLDB, Athens, Greece, 1997, pp. 156–165.
[4] V. Harinarayan, A. Rajaraman, J. Ullman, Implementing data cubes efficiently, in: Proc. ACM Sigmod Conf., Montreal, Canada, 1996.
[5] J. Yang, K. Karlaplem, Q. Li, Algorithms for materialized view design in data warehousing environments, in: Proc. 23rd VLDB, Athens, Greece, 1997, pp. 136–145.
[6] D. Theodoratos, M. Bouzeghoub, A general framework for the view selection problem for data warehouse design and evolution, in: Proc. DOLAP'00, Washington, DC, 2000, pp. 1–8.
[7] D. Munneke, K. Wahlstrom, M. Mohania, Fragmentation of multidimensional databases, in: Proc. 10th Australasian Database Conf., Auckland, 1999, pp. 153–164.
[8] M. Özsu, P. Valduriez, Principles of Distributed Database Systems, Prentice-Hall Int. Editors, 1991.

 [9] A. Datta, B. Moon, H. Thomas, A case for parallelism in data warehousing and OLAP, in: Proc. IEEE First Int. Workshop on Data Warehouse Design and OLAP Technology, 1998.
[10] M. Golfarelli, D. Maio, S. Rizzi, Applying vertical fragmentation techniques in logical design of multidimensional databases, in: Proc. DaWaK 2000, Greenwich, UK, 2000.
[11] V. Maniezzo, A. Carbonaro, M. Golfarelli, S. Rizzi, An ants algorithm for optimizing the materialization of fragmented views in data warehouses: Preliminary results, in: Lecture Notes in Computer Science, vol. 2037, Springer-Verlag, 2001, pp. 80–89.
[12] ILOG Inc., CPLEX 6.6 technical reference guide, 2001.
[13] M. Poess, C. Floyd, New TPC Benchmarks for decision support and web commerce, ACM SIGMOD Record 29 (4).
[14] J. Gray, A. Bosworth, A. Lyman, H. Pirahesh, Data-Cube: a relational aggregation operator generalizing group-by, cross-tab and sub-totals, Tech. Rep. MSR-TR-95-22, Microsoft Research, 1995.
[15] A. Gupta, V. Harinarayan, D. Quass, Aggregate-query processing in data-warehousing environments, in: Proc. 21rd VLDB, Zurich, Switzerland, 1995.
[16] M. Grötschel, R. Graham, L. Lovász, Handbook of Combinatorics, Elsevier, 1995.
[17] V. Maniezzo, A. Colorni, The Ant System applied to the quadratic assignment problem, IEEE Trans. Knowledge Data Eng. 11 (5) (1999) 769–778.
[18] S. Sait, H. Youssef, Iterative Computer Algorithms with Applications in Engineering, IEEE Computer Society Press, Piscataway, NJ, 1999.
[19] L. Wolsey, Integer Programming, Wiley-Interscience, 1998.
[20] M. Golfarelli, S. Rizzi, Wand: A case tool for data warehouse design, in: Demo Proc. 17th ICDE, Heidelberg, Germany, 2001.
[21] D. Theodoratos, S. Ligoudistianos, T. Sellis, View selection for designing the global data warehouse, Data Knowledge Eng. 39 (3) (2001) 219–240.
[22] W. Chu, I. Ieong, A transaction-based approach to vertical partitioning for relational database system, IEEE Trans. Software Eng. 19 (8) (1993) 804–812.
[23] X. Lin, M. Orlowska, Y. Zhang, A graph-based cluster approach for vertical partitioning in database design, Data Knowledge Eng. 11 (1993) 151–169.
[24] V. Gopalkrishnan, Q. Li, K. Karlapalem, Efficient query processing with associated horizontal class partitioning in an object relational data warehousing environment, in: Proc. DMDW'2000, Stockholm, 2000.
[25] P.E. O'Neil, D. Quass, Improved query performance with variant indexes, in: Proc. ACM Sigmod Conf., Tucson, Arizona, 1997, pp. 38–49.
[26] C.J. White, Sybase Adaptive Server IQ—A high-performance database for decision processing, Tech. rep., DataBase Associates International Inc., 1999.
[27] Sybase Inc., Adaptive Server IQ: Administration and performance Guide, 2001.
[28] A. Shukla, P. Deshpande, J. Naughton, K. Ramasamy, Storage estimation for multidimensional aggregates in the presence of hierarchies, in: Proc. 22nd VLDB, Mumbai, India, 1996, pp. 522–531.
[29] A. Cardenas, Analysis and performance of inverted database structures, Commun. ACM 18 (5) (1975) 253–263.

Matteo Golfarelli received his degree in Computer Science in 1995 and his Ph.D. for his work on autonomous agents in 1998 from the University of Bologna, Italy. He is Researcher at the Computer Science Department, and teaches Information Systems and Computer Architectures. He has published over 30 papers in refereed journals and international conferences in the fields of autonomous agents, pattern recognition and databases. He is currently involved in the EU PANDA thematic network concerning pattern-base management systems. His current research interests include all the aspects related to data warehouse design, in particular multi-dimensional modeling, view materialization and fragmentation, workload analysis.

Vittorio Maniezzo was born in Ferrara, Italy, and graduated in Electrical Engineering at the Polytechnic of Milan, where he also got his Ph.D. in Computer and Automation Engineering. After a post-doc at the Joint Research Centre of the European Commission dedicated to metaheuristics and decision support he won a position, currently as associate professor, at the Cesena site of the University of Bologna. His research activities are based on decision support systems and metaheuristics for combinatorial optimization. In particular, Vittorio Maniezzo contributed to the design of the ant system metaheuristic, of the Lagrangean column generation approach and published exact and heuristic solution methods for several problems, among which quadratic assignment, capacitated p-median, frequency assignment, project scheduling, capacitated arc routing.

Stefano Rizzi received his degree in Electronic Engineering in 1988 and his Ph.D. for his work on autonomous agents in 1996 from the University of Bologna, Italy. Currently he is Associate Professor at the University of Bologna, where is the head of the Data Warehousing Laboratory and teaches Advanced Information Systems and Software Engineering. He has published over 50 papers in refereed journals and international conferences, mainly in the fields of data warehousing, pattern recognition, mobile robotics, and multiagent systems. He joined several national research projects on the above areas and is currently involved in the EU PANDA thematic network concerning pattern-base management systems. His current research interests include all the aspects related to data warehouse design, in particular multidimensional modeling, view materialization and fragmentation, physical design.