How reduce the View Selection Problem through the CoDe Modeling

Valentina Indelli Pisano
Università Degli Studi Di Salerno
Itália

Michele Risi
Università Degli Studi Di Salerno
Itália

Genoveffa Tortora
Università Degli Studi Di Salerno
Itália

Abstract—Big Data visualization is not an easy task due to the sheer amount of information contained in data warehouses. Then the accuracy on data relationships in a representation becomes one of the most crucial aspects to perform business knowledge discovery. A tool that allows to model and visualize information relationships between data is CoDe, which by processing several queries on a data-mart, generates a visualization of such data. However on a large data warehouse, the computation of these queries increases the response time by the query complexity. A common approach to speed up data warehousing is precompute a set of materialized views, store in the warehouse and use them to compute the *workload queries*. The goal and the objectives of this paper are to present a new process exploiting the CoDe modeling through determining the minimal number of required OLAP queries and to mitigate the problem of view selection, i.e., select the optimal set of materialized views. In particular, the proposed process determines the minimal number of required OLAP queries, creates an ad hoc lattice structure to represent them, and selects on such structure the views to be materialized taking into account an heuristic based on the processing time cost and the view storage space. The results of an experiment on a real data warehouse show an improvement in the range of 36-98% with respect the approach that does not consider materialized views, and 7% wrt. an approach that exploits them. Moreover, we have shown how the results are affected by the lattice structure.

Index Terms—View Selection, Conceptual Modeling, OLAP Optimization.

1 Introduction

In the last years the use of decision support system based on data warehouse(DW) is widely increasing in different domains such as marketing, business research, demographic analysis, security and medical field. Whereby is necessary to analyze and solve several problems related data OLAP analysis (OnLine Analytical Processing) [1], such as big-data, performance, and data visualization. Since the large size of a DW and the OLAP queries complexity have considerably grown the query processing cost has a critical effect on the performance and the productivity of a decision support system. Moreover, performing queries frequently is expensive, producing wasted effort and making the data warehousing extremely slow.

A common approach to speed up data warehousing is precalculate materialized views computed on frequent data usage, and store them in the DW. This allows that the overall *workload queries* can be calculated starting from these views [2], [3], [4]. However this approach requires a set up phase that increases the costs, then there is the needs to use it until the most frequently OLAP queries are discovered (further increasing the set up time costs). To mitigate these issues, several greedy based algorithms have been proposed in the literature (e.g., [5], [6], [7], [8]). For example, Agrawal *et al.* [5], use a multidimensional lattice framework to determine a good set of views to materialize. However, the optimal solution requires to determine all the possible dependencies on the lattice. This operation is time-consuming, which raises when the size of the dataset increases, and is not applicable in practice. To mitigate this aspect, we only need to know a priori data and relationships among them to obtain the relevant information to precalculate. This information can be provided directly by the user through a high-level model defined by *CoDe* [9], [10] (see Fig. 1(a)).

CoDe is a visual language that allows to conceptually organize the visualization of reports. It models graphical representations involving more than one type of graph that have to be composed and aggregated through conceptual links showing relationships among data. CoDe adopts an hybrid modeling process combining two main methodologies: *user-driven* and *data-driven*. The first one aims to create a model according to the user knowledge, requirements and analysis needs, whilst the latter has in charge to concretize data and their relationships in the model through OLAP queries.

Different approaches have been presented to perform data visualization [11], [12]. They allows to view the data with different types of visual representations and to switch from a display to another, but they maintain the visualizations separated and not connected to each other.

In this paper, we present an optimization approach that exploits the CoDe modeling to mitigate the view selection problem (VSP). The proposed approach selects the minimal number of required OLAP queries, compacts them, and creates a lattice structure avoiding the explosion of the number of nodes. Moreover, heuristics and a minimum spanning tree (MST) algorithm have

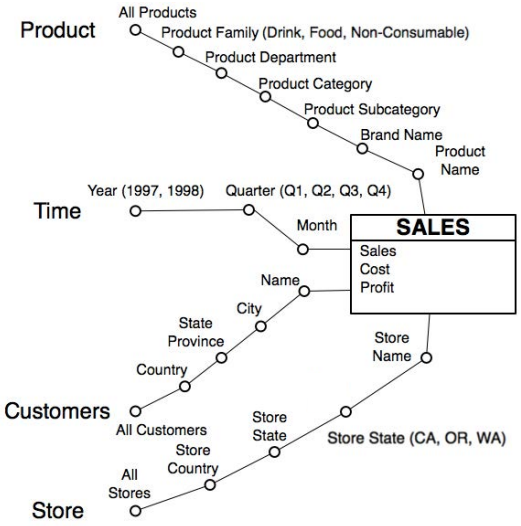


Fig. 2: Dimensional Fact Model of the *Sales* data-mart.

been adopted to select on the lattice the views to be materialized taking into account the processing cost and the view storage space. The proposed approach has been tested on the Foodmart DW [13] which maintains information about a franchising of big supermarkets located in the United States, Mexico and Canada. In particular, the data-mart *Sales* has been selected to analyze the sales of these stores, customers and products.

This paper extends our previous work published in [14] by:

- detailing the CoDe methodology and the visual notation;
- providing more details on the proposed process;
- enriching the related work discussion and comparison.
- evaluating the approach on another case study and on the whole CoDe model.

The paper is organized as follow. Section 2 outlines related works. Section 3 describes the optimization approach to select the materialized views and Section 4 discusses the results of a case study applied on two different CoDe models. Final remarks conclude the paper in Section 5.

2 Related Work

The VSP plays a central role in the design and query of a DW [1]. Many research was written to address this problem such as deterministic, genetics, and hybrid algorithms.

Harinarayan *et al.* [6] presented a greedy algorithm for the selection of materialized views using a constraint on the maximum number of views to materialize and a framework lattice to express dependencies between such views. Using the lattice structure some queries can be answered from the result of others optimizing the query processing costs. However, they did not take into account the storage space constraints. Our first goal was to develop a framework lattice more scalable than theirs to select the views taking into account this information and making our solution closer similar to the real problem.

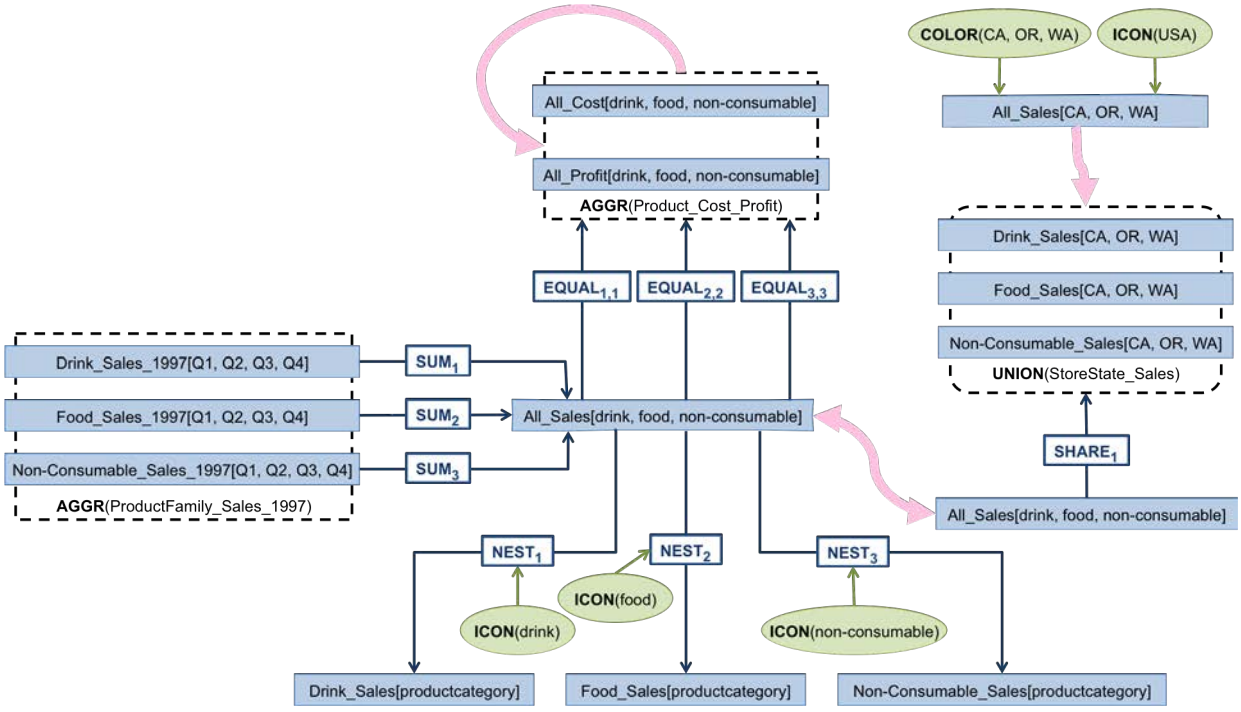
Shukla *et al.* [7] addressed the VSP by exploiting a greedy algorithm which picks the views by focusing on a benefit metric, such metric is based on the probability which each view being queried. However, this solution is not suitable because it needs to know a priori the frequency with which the view is expected to be queried.

Yang *et al.* [15] proposed a heuristic algorithm which utilizes a Multiple View Processing Plan to obtain an optimal materialized view selection. They use a tree structure which every node is a potential candidate to the materialization. If a node is considered a good candidate, the savings will calculate taking into account the materialization costs and subtracting the cost maintenance. If the result is a positive value, the node is added to the tree, otherwise it is deleted with all its descendants from the candidate set. In this way the best combination of performance and maintenance costs can be achieved. However, this algorithm did not consider the storage space constraints.

Many genetic solutions were proposed, such as [16], where the views are chosen in terms of reduction on the processing costs and maintenance costs. However, for the random characteristics, this algorithms do not give an optimal solution. Another approach is represented by the hybrid algorithms. Zhang *et al.* [8] applied their hybrid approach combining greedy and genetic algorithms to solve three types of problems. The first one addressed the optimization of queries, the second concerned the choice of the best execution plan for each query and the third was about the views selection problem. However, such algorithms are characterized on an high computational complexity.

Differently, other approaches exploit clustering algorithms. In particular, they define a similarity function which makes sure that similar objects belongs to the same group.

Gong *et al.* [17] defined an algorithm that firstly clusters materialized views, and then dynamically adjusts them. They propose a similarity function that is a weighted sum between the query processing cost and the maintenance cost of view. Their experimental results show that the algorithm not only improves the overall query response performance, but also reduces the computational cost which will be spent during the update of the materialized view. Similarly, Chaudhari *et al.* [18] defined an algorithm that clusters materialized views in many steps. The first step removes the materialized views with both low access frequency and high storage space for the materialization of new views.The second step selects queries based on their weight age in the given query set and storage space. The queries which have high access frequency are selected for the view selection problem. Then third step assigns an integer value to each query adopting some predefined criteria. The



(a)

(b)

Fig. 1: CoDe model for the data-mart *Sales* (a), and its graphical representation (b).

four step calculates for each view the query processing and the maintenance costs. Finally, the total cost of the materialization is computed by summing the query processing and maintenance costs. Such algorithms achieve good performance in the optimization of the query cost, but they require to manually select a threshold value.

3 CoDe Modeling and VSP optimization

The optimization approach is composed of three phases:

A. *Code Modeling*. It produces as output a model describing information items and their relationships. This phase is performed by the company manager that is the expert of the specific domain.

B. *OLAP Operation Pattern Definition*. It is used to define the sequences of operations needed to extract all the information.
C. *OLAP Operation Optimization*. In order to speed-up the data extraction, this phase selects the set of views to be materialized and maps the OLAP operation patterns into OLAP queries, which are used to extract information from the data-mart.

The extracted information is used to display the final report taking into account data series and their relationships [19] according to the CoDe model.

The Dimensional Fact Model of the Sales data-mart is shows in Fig. 2. It consists of a fact schema SALES with measures Sales, Cost, and Profit, and dimensions Customers, Store, Product and Time. The maximum level of aggregation (named ALL_{dw}) is represented by ALL for Products, ALL for Stores ed ALL for Customers, otherwise in the absence of an ALL level, as for the Time dimension, the members at the top are all those contained in the Year level.

3.1 Code Modeling

The CoDe Modeling phase produces a CoDe model (see Fig. 1(a)), which generates the report shown in Fig. 1(b). Such model is composed by *Terms*, *Functions* and *Relations* [9].

A *Term* is an array of components, is identified by a name and has associated data extracted from data-mart. In Fig. 3, components in the square brackets represent members or dimensional attributes, whilst the name can specify measures and/or hierarchies. In particular, the name All_Sales indicates the measure Sales with the maximum level of aggregation, and the three components denotes the members belonging to the dimensional attribute *Product Family*. Practically, it corresponds to the total sales for the three members.

A *Function* is adopted to link terms provided as input by defining constraints and correspondences among their components. The function $AGGREGATION$ is used to group several terms having the same components into a single term preserving the original data values for each component. The output term includes both the involved terms and the AGGR label. At the right side of the Fig. 1a we find such function (with label ProductFamily_Sales_1997) that allows to group the sales of the drink, food and non-consumable components for the four quarters in the year 1997.

The function SUM_i has two input terms. As a precondition the value of i-th component in the second term is the sum of the data series in the first one. In the example three function SUM_1, SUM_2 and SUM_3 associate the sum of all the values of data series drink, food and non-consumable to the term All_Sales.

All_Sales[drink, food, non-consumable]

Fig. 3: Definition of the term All_Sales.

OLAP operation pattern for the *CoDe Term*
$pivot(h \cup v; m)^*[rollup(h; m)\|drilldown(h; m)]^*$
$dice(h; m)[rollup(v; m)\|drilldown(v; m)]^* slice(v; m)$

Fig. 4: The operation pattern for the *CoDe Term*.

The function $NEST_i$ has a symmetric definition with respect to the SUM_i function. It applies to two input terms where one component in a report has a value aggregated from data in the other one. At the bottom side of the Fig. 1a is represented such function, in this example the term All_Sales is given in input to the functions $NEST_1$, $NEST_2$ and $NEST_3$ in order to detail the sales of its three components drinks, food and non-consumable distinct by Product Category. The details are represented by the Drink_Sales, Food_Sales and Non-Consumable_Sales terms. In addition, to each nest function is applied $ICON$ that adds an icon in the generated report. The three terms Drink_Sales, Food_Sales and Non-Consumable_Sales on the right side of Fig. 1a represent the data series (distinct by product family) containing the values of total incomes made in the stores of three different American states, such as California (CA), Oregon (OR) and Washington (WA).

The $UNION$ function is the same as the aggregation function, but different sets of components are allowed in the input terms. The representation of the output CoDe term is denoted by the UNION label. In the example in Fig. 1a, the three reports Drink_Sales, Food_Sales and Non-Consumable_Sales has been aggregated in the term StoreState_Sales by using this function.

The function $SHARE_i$ shares all the n components of the input term on the i-th position of the other n output ones, respectively. In the example in Fig. 1a this function is defined between All_Sales and StoreState_Sales.

A *Relation* is a logic connection existing between two terms. An example is shown in the top right part of Fig. 1a where the term *All_Sales(CA, OR, WA)* is related with the term *StoreState_Sales* through a thick arrow. The first term represents the report relative to the total receipts of the sales made on any family of products from the shops located in the CA, OR and WA states. The output is a graphical representation of the input terms augmented with links among correspondences of each couple of components. Moreover, $ICON$ and $COLOR$ are applied on All_Sales(CA, OR, WA). In particular, the ICON(USA) displays in the final visualization the total incomes of the sales on the United States geographical map, and COLOR(CA, OR, WA) highlights the three states with different colours on such map.

The two terms *All_Profit* and *All_Cost* represent profits and costs respect with the total sales for each product family. These data series are aggregated with the $AGGREGATION$ function, which groups them for each product family. The recursive relation applied on the aggregation, adds to the final visualization the total incomes related to the sales for the product families

drink, food and non-consumable.

The function $EQUAL_i$ has two input terms $T_1[D_1, ..., D_i, ..., D_h]$ and $T_2[C_1, ..., C_j, ..., C_n]$, where the i-th component of term T_1 is equal and has the same associated value of the j-th component of T_2. At the top side of the Fig. 1a the three functions $EQUAL_{1,1}$, $EQUAL_{2,2}$, and $EQUAL_{3,3}$ and the $AGGREGATION$ named *Product_Cost_Profit*, define an identity between the value of the i-th component of All_Sales and the i-th component given in output by the recursive $Relation$ applied on the $AGGREGATION$ function ($\forall i = 1, 2, 3$).

Finally, the bidirectional relation between the two terms *All_Sales* produces two different visual representations and one-to-one graphical links among the their components. Further details on the CoDe visual language syntax are available in [9].

The data series are extracted from the data-mart by applying a combination of selection and/or aggregation dimensional operators, i.e., the OLAP operations that allow multidimensional data analysis: *slice/dice/pivot/rollup/drilldown* [20]. Thus, all the syntax constructs of CoDe are expressed as a combination of them, named *Operation Patterns* [9].

The operation pattern to extract information for any *CoDe Term* is shown in the first row of Table 1, where each label represents a single OLAP operation, whilst the symbol h (resp. v) in the parentheses denotes the horizontal (resp. vertical) axis on which the operation is performed (the multiplicity is expressed by the $*$ symbol). The parameter m after the symbol ; represents the set of members (separated by the comma) on which the OLAP operation is performed. In particular, $pivot(h \cup v; m)$ is used to rotate the dimensional members m on any single dimension, $rollup(h; m)/rollup(v; m)$ or $drilldown(h; m)/drilldown(v; m)$ are performed on the horizontal/vertical axis in order to decrease or increase the details of the set m, respectively, $dice(h; m)$ is performed on the horizontal axis to select a subset of dimensional members m and to exclude the others (if present), and $slice(v; m)$ is executed on the vertical axis to reduce the number of selected dimensional members to the ones in m.

3.2 OLAP Operation Pattern Definition

This phase takes in input the CoDe model, computes the sequence of operation patterns and then provides as output a lattice representing the set of candidate views to materialize in the DW. Such phase is composed of three steps (see Fig. 5).

- *Eligible patterns generation.* Starting from the CoDe model, this phase selects the attributes in the DFM on which a finite sequence of OLAP operations has to be executed (named *eligible pattern*). The eligible pattern is determined taking into account the OLAP operation patterns for each term and for every term derived from the functions and/or relations in the CoDe model. Two examples of eligible pattern generation for the $CoDe\ Term$

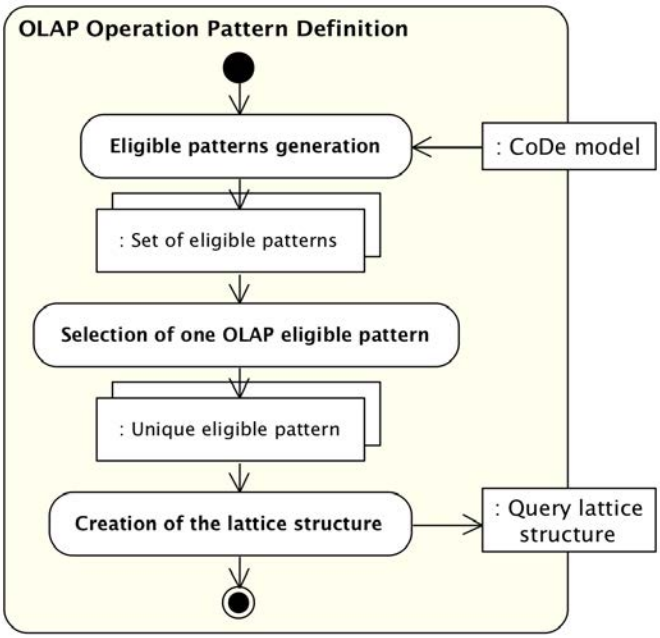


Fig. 5: The OLAP Operation Pattern Definition phase.

and for the SUM_i function are provided in Algorithm 1 and Algorithm 2, respectively. The Algorithm 1 generates the eligible pattern for the $CoDe\ Term$. In particular, it computes the set of attributes to perform the OLAP operation by exploiting the input/output attributes of a term (lines 2-9), while at line 10 the eligible pattern is created by following the OLAP pattern defined for that term. The symbol ! denotes that the OLAP operation produces the same result independently from the order of dimensional members on which it is applied, while the symbol $\cup$ denotes the operations that can be executed in any order. Finally, at lines 11-12 the computed dimensional members and the eligible pattern are associated to the term. Similarly, the Algorithm 2 computes the set of attributes needed to perform the OLAP operations by exploiting the input/output attributes of the terms involved in the SUM function.

- *Selection of one OLAP eligible pattern.* The previous phase generates a set of eligible patterns for each term, function and relation in the CoDe model. This phase adopts the heuristic strategy to select one eligible pattern from each set. The selected one is the longest common prefix (LCP) of OLAP operations also considering their permutation. This strategy is implemented in the Algorithm 3. In order to simplify the description of the algorithm, we represent each OLAP operation in the eligible pattern with a unique label. For example, let the eligible pattern $[pivot(h; m_1, m_2, ..., m_n)]!slice(v; m_k)$ and the labels $a = pivot(h; m_1)$, $b = pivot(h; m_2)$, $c = pivot(h; m_n)$, $d = slice(v; m_k)$, thus, the eligible pattern is represented as the string $\{a, b, c\}, d$ where the a, b, c labels can be swapped.

The Algorithm 3 builds a prefix tree and adopts a greedy strategy to selects the prefixes, through a

TABLE 1: Summary of the mapping among Code syntax and OLAP operation patterns.

CoDe syntax	OLAP operation pattern
$CoDe\ Term$	$pivot(h \cup v)^*[rollup(h)\|drilldown(h)]^*dice(h)[rollup(v)\|drilldown(v)]^*slice(v)$
SUM_i	$rollup(h)^*pivot(h_i \cup v)dice(h_1 \dots h_i \dots h_n)$
$NEST_i$	$slice(h_1 \dots h_{i-1}h_{i+1} \dots h_n)drilldown(h_i)^*$
$EQUAL_{ij}$	$slice(h_1 \dots h_{i-1}h_{i+1} \dots h_n)pivot(h_j \cup v)dice(h_1 \dots h_j \dots h_m)$
$SHARE_i$	$\forall j = 1, \dots, n \quad slice(h_1 \dots h_{j-1}h_{j+1} \dots h_n)pivot(h_i \cup v)\|dice(h_1 \dots h_i \dots h_m)$
AGGREGATION / UNION	$pivot(h \cup v)^*[rollup(h)\|drilldown(h)]^*dice(h)\ [rollup(v)\|drilldown(v)]^*slice(v)$
RELATION	$\forall j = 1, \dots, n \quad slice(h_1 \dots h_{j-1}h_{j+1} \dots h_n)\ drilldown(h_i)^*$

To ease the readability the OLAP operation patterns, we have omitted the set of members on which the OLAP operations is performed.

breadth-first search on the maximum number of strings exchangeable that share a prefix, and a depth-search on the maximum length of the common prefixes. The LCP algorithm is used by Algorithm 4 to determine one of OLAP eligible patterns, split it into a sequence of OLAP operations, associate a unique label to each single OLAP operation, and finally select the longest common prefix. As an example, given the sets of eligible patterns: $S_1 = \{a, c\}; S_2 = \{a, b, c\}; S_3 = \{a, d\}, e, f; S_4 = \{a, d\}, f; S_5 = \{a, d\}, e$. The algorithm *LCP* generates the prefix tree in Fig. 6, obtaining the set of unique eligible patterns $S_1 = a, c; S_2 = a, c, b; S_3 = a, d, e, f; S_4 = a, d, e; S_5 = a, d, f$.

Algorithm 1 Eligible pattern for the $CoDe\ Term$.

Require: The CoDe model
1. **for all** $term$ in the CoDe model **do**
2. $cols_1$={set of measures: Sales, Cost, Profit}
3. $rows_1$={set of dimensional members that belongs to ALL_{dw}}
4. $cols_2$={set of dimensional members used to compute the column attributes of the $term$ (i.e., components)}
5. $rows_2$={set of measures/hierarchies used to compute the row attributes of the $term$}
6. $pivoting_{v/h}$={set of dimensional members of $cols_1/rows_1$ present in $rows_2/cols_2$ to compute the $vertical/horizontal$ pivoting}
7. $slicing_v$={set of dimensional members of $rows_1$ not present in $rows_2$ to compute the vertical slicing}
8. $dicing_h$={set of dimensional members used to compute the col attributes of the $term$}
9. $drilling_{h/v}$={set of dimensional members represented by ancestors recursively computed of the dimensional members in $cols_2/rows_2$ present in $cols_1/rows_1$}
10. $eligible$ = $[pivot(h; pivoting_h) \cup pivot(v; pivoting_v)]!$ $[drilldown(h; drilling_h)]![dice(h; dicing_h)]$ $[drilldown(v; drilling_v)]![slice(v; slicing_v)]!$
11. $term.rows/cols = eligible.rows/cols$
12. $term.eligible\_pattern = eligible.olap\_pattern$
13. **end for**

Algorithm 2 Eligible pattern for the SUM_i function.

Require: The SUM_i function between the T_1 and T_2 terms in the CoDe model
1. $cols_1/rows_1$={set of dimensional members used to compute $T_1.cols/T_1.rows$}
2. $cols_2/rows_2$={set of dimensional members used to compute $T_2.cols/T_1.rows$}
3. $rolling_h$={set of dimensional members of $cols_1$ to be aggregated to obtain the set of attributes of $cols_2$}
4. $pivoting_i$={the i-th attribute of $rows_2$}
5. $pivoting_v$={set of dimensional members of $rows_1$}
6. $dicing_h$={set of dimensional members of $cols_2$}
7. $eligible$ = $[rolling(h; rolling_h)]!$ $[pivoting(h; pivoting_i) \cup pivoting(v; pivoting_v)]!$ $[dicing(h; dicing_h)]$
8. $SUM_i.eligible\_pattern = eligible.olap\_pattern$

Algorithm 3 LCP(S, root).

Require: $S = \{s_1, s_2, \dots, s_n\}$, root
1. **for all** s_i in S with 0 characters **do**
2. addLabelNode(root, "s_i")
3. remove s_i from S
4. **end for**
5. **if** $|S| == 1$ **then**
6. addLabelNode(root, "s_1"(s_1))
7. remove s_1 from S
8. **end if**
9. **if** $|S| == 0$ **then**
10. **return**
11. **end if**
12. V = sort(unique_characters(S))
13. i = 1
14. **while** $|S| \geq 1$ **do**
15. c = $V[i]$
16. S_c = select strings in S starting with c
17. remove S_c from S
18. **if** $|S_c| > 1$ **then**
19. node=createChild(root)
20. setLabelEdge(root, node, c)
21. **else**
22. node=root
23. **end if**
24. remove the first character c from the strings in S_c
25. *LCP*(S_c, node)
26. i++
27. **end while**
28. **return**

Algorithm 4 Selection of OLAP eligible patterns.

Require: Set E of all eligible patterns
1. map each OLAP operations in E with an unique character
2. root = create a new node
3. *LCP*(E, root)
4. **for all** string $s_i \in E$ **do**
5. prefix = concatenation of characters on the path from the root to the node labeled with s_i
6. remove from the selected string s_i the characters in *prefix*
7. build the unique OLAP eligible pattern by concatenating prefix with the string s_i
8. **end for**

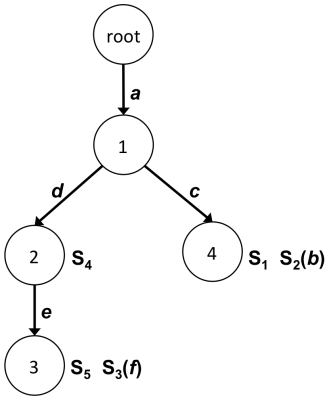


Fig. 6: The prefix tree.

- *Creation of the lattice structure.* The lattice structure is created by using the OLAP eligible patterns. The lattice is a directed acyclic graph and is defined as follow: *i)* Each node in the lattice represents a view that has to be computed on the DW. *ii)* Let u and v two views, each edge between u and v represents an OLAP operation that applied on u computes v. *iii)* There exists a partial order $\preceq$ between views in the lattice: $v \preceq u$ if and only if v can be computed starting from *u (dependency)*. *iv)* There is the aggregated view "ALL" in the lattice, upon which every view is computed. The ancestors of v is the set $\{s|s \preceq v\}$. Thus v is computed starting from any of its ancestors, i.e., the views it transitively depends on, applying the sequence of OLAP operations specified on the edges among any s and v. To tackle the state-space explosion problem [21], the views of the lattice structure are merged by exploiting the common prefixes of unique eligible patterns.

To define the processing cost of a view, we compute a cost model by considering the processing cost of each single OLAP operation that produces that view. This processing cost (i.e., P_c) is a linear function applied on the view size (i.e., V_s) and it is expressd by the formula $P_c = m_o * V_s + o$, where m_o is a multiply coefficient depending on the OLAP operation and o is a fixed cost

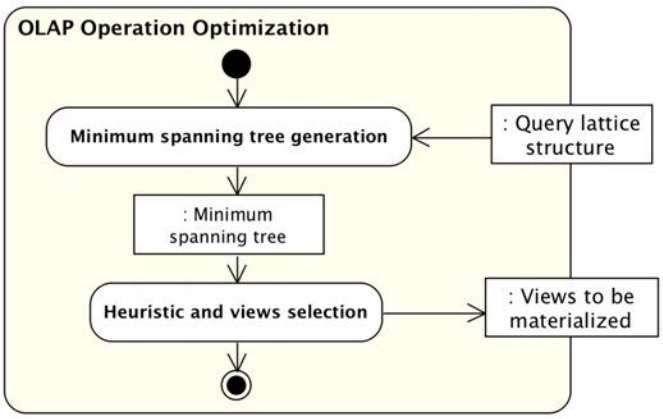


Fig. 7: The OLAP Operation Optimization phase.

TABLE 2: Coefficients to compute V_s and P_c.

OLAP Operation	$\mathbf{Rows_v}$	$\mathbf{Cols_v}$	Coefficient $\mathbf{m_o}$
pivot(h) or pivot(v)	$rows$	$cols$	$2.5 * 10^{-3}$
slice(h)	1	$cols$	$5.0 * 10^{-3}$
slice(v)	$rows$	1	
dice(h)	$rows - 1$	$cols$	
dice(v)	$rows$	$cols - 1$	
rollup(h)	$rows/2$	$cols$	$10.0 * 10^{-3}$
rollup(v)	$rows$	$cols/2$	
drilldown(h)	$2 * rows$	$cols$	
drilldown(v)	$rows$	$2 * cols$	

The coefficient o is $5 * 10^{-3}$ and it is fixed for all the OLAP operations (m_o and o are expressed in secs.).

(e.g., the overhead of running a query on a negligible DW size). These two coefficients have been empirically determined by executing all the five OLAP operations on different DWs. The view size V_s is computed by multiplying the $Rows_v$ and $Cols_v$ coefficients (i.e., the number of rows and the columns obtained from the OLAP operation). Table 2 summarizes the coefficients for computing the view size and processing costs in terms of OLAP operations.

3.3 OLAP Operation Optimization

This phase takes as input the lattice and gives as output the set of views to be materialized. Figure 7 details the two steps composing this phase.

- *Minimum spanning tree generation.* The problem of view selection (*VSP*) is known to be NP-complete by a reduction from minimum set cover [22] and it can be formulated as follows. Given a query workload $Q = \{q_1, q_2, \ldots, q_q\}$ defined over the lattice L composed of n nodes (i.e., $v_1, \ldots, v_n$), the problem is to select an appropriate set of views to materialize $M = \{mv_1, mv_2, \ldots, mv_m\}$ (with $m \leq n$) such that Q is answered starting from M with the lowest processing cost under a limited amount of resources, e.g., storage space [23]. Since the lattice structure does not have an unique path from the root to each node, a MST is computed to select the paths that starting from the root node generate the views (answering the workload query Q) with the lowest processing costs. This is performed by considering as weights the set of P_c required to compute OLAP operations among each couple of directly connected nodes in the lattice. Then, each $node$ in the MST has associated a couple of weights that represent the view processing cost and the view size. These values are computed as follows: *i)* $T(root, node)$ is the processing cost of all the OLAP operations from the root to $node$, and is the given by:

$$\sum_{op \in path(root, node)} P_c(op) \quad (1)$$

where $path$ computes the sequence of OLAP operations among two nodes in the MST. *ii)* Starting from the rows

and columns of the view represented by the root of MST, $S(node)$ is computed as the product of $Rows_{node}$ and $Cols_{node}$. These values are updated taking into account the OLAP operations present in the path from root to $node$ and coefficients in Table 2.

- *Heuristic and views selection.* On the MST, the *VSP* can be reduced to an optimization problem where we are interested to minimize the following objective function:

$$\min \sum_{q_j \in Q} T(mv_r, v_j) \tag{2}$$

$$where \sum_{mv_i \in M} Size(mv_i) \leq S \quad and \quad r \in 1, \ldots, m$$

and $T(mv_r, v_j)$ is the processing cost of the view v_j that answers the query q_j, starting from the materialized view mv_r and *S* is the available total storage space. The problem can be approached by following two steps: *1)* Calculate the processing cost and the view size for each node in MST where $M = \emptyset$ (i.e., no materialization has been computed); *2)* Add in M the set of nodes present in the MST that minimizes the objective function and respects the space constraints [14].

Many algorithms has been proposed to select properly the set M [5], [6], [16], [24], [25], [26]. Most of all focused on the concept of a *benefit* metric and differ from each others in the definition of this metric. Informally, the benefit to materialize a view is the savings we obtain choosing to materialize such view instead of another one. Given an instance of the VSP problem, Shukla *et al.* [7] introduce a benefit metric called *Average Query Cost* ($AvQC$) computed as follow:

$$\sum_{j=1}^{n} p_j \cdot T(mv_j, v_j) \tag{3}$$

where $p_1, \ldots, p_n$ represent the probabilities that queries $q_1, \ldots, q_n$ occur. These queries are answered by the views $v_1, \ldots, v_n$, starting from the materialized view $mv_1, \ldots, mv_n$. This approach differs from ours because the authors want to minimize the ratio between the size of the query to pick up, and the probability of its occurrence. In the case the materialized views have not an equal probability of being queried, the user has to assign such probabilities. However, the frequency (i.e., the probability) a materialized view is expected to be queried is not always a priori known, and this frequency may change during the DW process.

Algorithm 5 HRU_T

Require: The MST, k
1. $M = \{root\}$
2. **for** $i = 0$ **to** k **do**
3. **if** $\exists v \in MST \backslash M$ that maximizes $Inv(v, M)$ **then**
4. $M = M \cup v$
5. **end if**
6. **end for**
7. **return** M

Differently, Harinarayan *at al.* define in [6] two different benefit metrics taking into account the processing cost and the space occupied by the materialized views. The first one is defined as follow and is used in the Algorithm 5.

Let v a view in the MST, for each view $w \preceq v$ (i.e., w in the MST that covers v):

$$Inv(v, M) = \sum_{w \preceq v} I(v, w, M) \tag{4}$$

$$where\ I(v, w, M) = \begin{cases} T(u, w) - T(v, w) \\ \quad if\ T(u, w) > T(v, w) \\ 0 \quad otherwise \end{cases}$$

and u is the materialized view in M with the lowest cost that is covered by w. Summarizing, the cost of evaluating w by using v is compared wrt. the cost of evaluating w by using a materialized view u. If v helps (i.e., the cost of v is less than the cost of u), then the difference represents part of the benefit of v in case it is selected as a materialized view. The total benefit $Inv(v, M)$ is the sum over all views that cover v. The algorithm HRU_T maximizes the benefit (line 3), by adding the selected view in the set M (line 4) until the fixed limit (i.e., k) on the number of view to materialize is reached. At line 7 the algorithm returns the set M containing the selected view to materialize.

The second benefit metric, is defined as follow and is used in the Algorithm 6.

$$InvS(v, M) = \frac{Inv(v, M)}{Size(v)} \tag{5}$$

The metric considers the view space occupied by M, and is calculated as the ratio between the investment to compute v and all its descendant, and the space to materialize it. The algorithm HRU_S maximizes the total benefit (line 3), by adding the view v in M (line 4) as long as the upper-bound of the disk space (i.e., s) is not been reached. The algorithm does not consider the space occupied by the root which is always materialized. At line 8 the algorithm returns the set M containing the selected view to materialize.

4 Case Study

In this case study, to simplify the presentation we show the optimization process on a portion of the CoDe model

Algorithm 6 HRU_S

Require: The MST, s
1. $M = \{root\}$
2. **while** $s > 0$ **do**
3. **if** $\exists v \in MST \backslash M$ that maximizes $Inv(v, M)$ and $s - Size(v) > 0$ **then**
4. $M = M \cup v$
5. $s = s - Size(v)$
6. **end if**
7. **end while**
8. **return** M

TABLE 3: Eligible OLAP operation patterns for the term *Drink*.

1)	[pivot(h; [Time].[1997],[Time].[1998])]!
2)	drilldown(h; [Time].[1997])
3)	dice(h; [Measures].[Sales], [Time].[1997].[Q1], [Time].[1997].[Q2], [Time].[1997].[Q3], [Time].[1997].[Q4])
4)	drilldown(v; [Product].[All Products])
5)	[slice(v; [Store].[All Stores], [Customers].[All Customers], [Product].[All Products].[Food], [Product].[All Products].[Not-Consumable])]!

TABLE 4: Eligible OLAP operation patterns for SUM_1.

1)	[rollup(h; [Time].[1997].[Q1], [Time].[1997].[Q2], [Time].[1997].[Q3],[Time].[1997].[Q4])]!
2)	[pivot(v; [Time].[1997])∪ pivot(h; [Product].[All Products].[Drink])]!
3)	dice(h; [Measures].[Sales], [Product].[All Products].[Drink], [Product].[All Products].[Food], [Product].[All Products].[Non-Consumable])

of Fig. 1(a), i.e., *AGGR(ProductFamily_Sales_1997)* and the term *All_Sales*. In the CoDe model the terms *Drink*, *Food* and *Non-consumable* representing the data series for each product family sold in the Foodmart stores which are respectively drinks, food products and not edible products. These data series are referred to the sales in four quarters (i.e., *Q1*, *Q2*, *Q3*, *Q4*) of the 1997, and on them is applied an aggregation function (i.e., *AGGR*) that allows to group the sales of drink, food and non-consumable for four quarters. The *All_Sales* term represents the cumulative data series of the total sales made in the 1997 for each product family, and the SUM_1, SUM_2, and SUM_3 functions are used to map the sum of data series *Drink*, *Food* and *Non-consumable*, respectively.

The first step of the *OLAP Operation Pattern Definition* phase generates the eligible patterns for the four terms *Drink*, *Food*, *Non-Consumable*, *All_Sales*, and for the functions *AGGR ProductFamily_Sales_1997*, SUM_1, SUM_2, and SUM_3. Table 3 show the outputs obtained for the term *Drink*. The second step selects the unique OLAP patterns, decomposing the OLAP operation patterns of each dimensional members and by renaming them with a unique labels. The output is a vocabulary table shown in Table 5. The set *Ss* of switchable strings replacing each OLAP operation with a label is built, and for each switchable string in *Ss*, the OLAP operation patterns is computed by following the path on the prefix tree (see Table 6). However since the *SUM* functions do not share the first input term then the OLAP unique operation, patterns are casually computed respecting the dimensional operation order. The output of this step is shown in the right side part of the Table 6. The last step aims to build the lattice structure (see Fig. 8). In such structure, the edges represent OLAP operations, the nodes correspond to the generated views, and each view has a path which starts from the root. The *OLAP Operation Optimization* phase generates the MST from the lattice structure. Figure 8 shows the MST with the view space and the processing cost computed for each node in according to the proposed cost model, the nodes colored in gray represent the views of the workload. Once generated the MST, the views to materialize by applying the solution proposed by [6], [7] are selected. The HRU_T procedure (with $k = 3$) selects the views: $root$, v_3, v_9 and v_{18}. The HRU_S procedure with storage space $s = 81$ (given multiplying 3 by the average size of views in the MST), selects the views: $root$, v_9, v_{11}, v_{23}. Finally, the algorithm proposed by Shukla *et al.* [7], by assuming that all aggregates have an equal probability of being queried, selects the views: $root$, v_{10}, v_{23}, v_{24}.

TABLE 5: Vocabulary table.

a =	pivot(h; [Time].[1997])
b =	pivot(h; [Time].[1998])
c =	drilldown(h; [Time].[1997])
d =	dice(h; [Measures].[Sales])
e =	dice(h; [Time].[1997].[Q1])
f =	dice(h; [Time].[1997].[Q2])
g =	dice(h; [Time].[1997].[Q3])
h =	dice(h; [Time].[1997].[Q4])
i =	drilldown(v; [Product].[All Products])
j =	slice(v; [Store].[All Stores])
k =	slice(v; [Customers].[All Customers])
x =	rollup(h; [Time].[1997].[Q4])
l =	slice(v; [Product].[All Products].[Food])
m =	slice(v; [Product].[All Products].[Non-Consumable])
n =	slice(v; [Product].[All Products].[Drink])
o =	pivot(h; [Product].[All Products])
p =	drilldown(h; [Product].[All Products])
q =	dice(h; [Product].[All Products].[Drink])
r =	dice(h; [Product].[All Products].[Food])
s =	dice(h; [Product].[All Products].[Non-Consumable])
t =	slice(v; [Time].[1998])
u =	rollup(h; [Time].[1997].[Q1])
v =	rollup(h; [Time].[1997].[Q2])
w =	rollup(h; [Time].[1997].[Q3])
y =	pivot(v; [Time].[1997])
z =	pivot(h; [Product].[All Products].[Food])
$\boldsymbol{\alpha}$ =	pivot(h; [Product].[All Products].[Drink])
$\boldsymbol{\beta}$ =	pivot(h; [Product].[All Products].[Non-Consumable])

TABLE 6: Switchable strings and OLAP unique operation patterns.

S_1-Drink =	{a, b} c d e f g h i {j, k, l, m}	a b c d e f g h i j k l m
S_2-Food =	{b, a} c d e f g h i {n, m, j, k}	a b c d e f g h i j k m n
S_3-Non-Consumable =	{b, a} c d e f g h i {j, n, k, l}	a b c d e f g h i j k l n
S_4-All Sales =	o p d q r s {t, j, k}	o p d q r s t j k
S_5-AGGR =	{a, b} c d e f g h i {j, k}	a b c d e f g h i j k
S_6-SUM$_1$ =	{u, v, w ,x} {y, α} d q r s	u v w x y α d q r s
S_7-SUM$_2$ =	{u, v, w, x} {y, z} d q r s	u v w x y z d q r s
S_8-SUM$_3$ =	{u ,v ,w ,x} {y, β} d q r s	u v w x y β d q r s

We have evaluated the processing time of each operation and the storage space value with the Saiku 2.4 suite [27] installed on laptop with a 2.93 GHz i3 processor, 4 GB of RAM and Windows 7. To mitigate the caching effect during the execution of multiple queries, we cleaned the memory cache before of each execution.

Table 7 summarizes the results of the three algorithms adopted in this paper. In particular, *Size(M)* indicates the size occupied by the materialized views, *Time(V, M)* indicates the total processing time to produce all the views *V* in the MST, whilst *Time(Q, M)* is the processing time of the materialized views that answer to the workload queries. The algorithm HRU_S reduces the total processing time to answer the workload queries and the

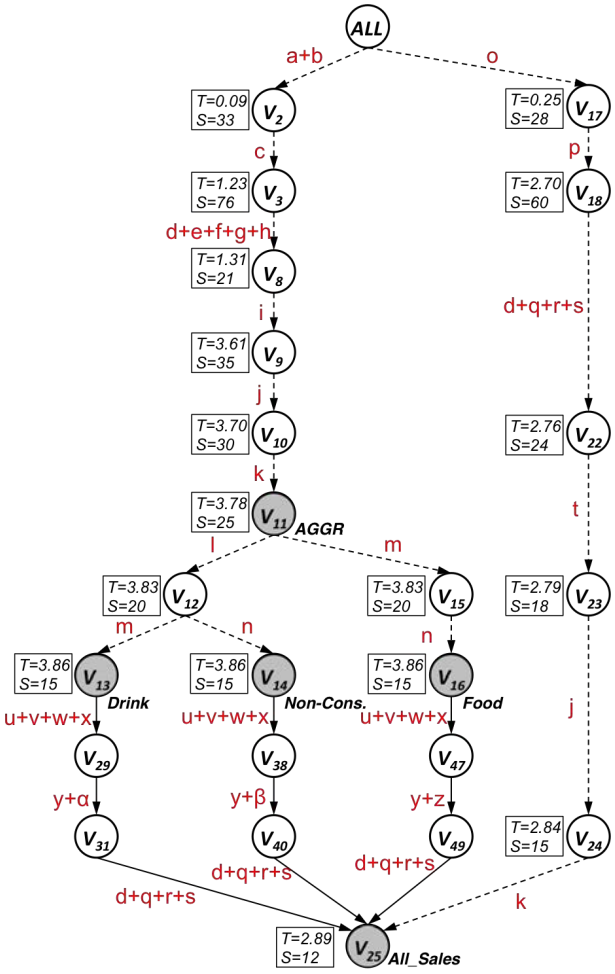


Fig. 8: The lattice structure with the OLAP operations and the corresponding MST (dashed arrows).

used storage space (78MB wrt. HRU_T that uses 171MB). On the contrary, HRU_T has a better performance in term of processing time when all the views have to be calculated. Moreover, $AvQC$ has similar results in term of storage space and worst processing time wrt. HRU_S.

Table 8 shows the results of the three algorithms when two consecutive executions are performed. In particular, the processing times are improved in the range of 34-36% in the second execution. Moreover, HRU_S gives better performance than the HRU_T, reducing its processing time to 0.34 seconds. The overall improvement of HRU_S is 62% after the first execution and 98% after the second one, comparing with the algorithm that does not materialize views. The improvement of HRU_S is of 5% wrt. the other two algorithms.

To demonstrate the scalability of the proposed process on the entire *Sales* data-mart, we have used the whole CoDe model in Fig. 1(a) obtaining comparable results. Indeed, the algorithm HRU_S (with $s = 180$) reaches an improvement of 60% wrt the algorithm that does not materialize views and reduces the processing time obtaining an improvement of 9% with respect the algorithm HRU_T exploiting the materialized views when

TABLE 7: Processing time and storage space of adopted algorithms applied on the lattice structure of Fig. 8.

	No Mat.	**AvQC**	**HRU$_T$**	**HRU$_S$**
Selected views	$root$	$root$, v_{10}, v_{23}, v_{24}	$root$, v_3, v_9, v_{18}	$root$, v_9, v_{11}, v_{23}
Size(M)	-	66MB	171MB	78MB
Time(V, M)	47.19s	16.55s	8.66s	15.49s
Time(Q, M)	18.25s	7.38s	7.42s	6.91s

TABLE 8: Comparison of processing times of two consecutive executions.

	No Mat.	**AvQC**	**HRU$_T$**	**HRU$_S$**
1st execution-Time(Q, M)	18.25s	7.38s	7.42s	6.91s
1st execution-Profit	-	59%	59%	62%
2nd execution-Time(Q, M)	18.25s	1.09s	1.11s	0.34s
2nd execution-Profit	-	93%	93%	98%

TABLE 9: Algorithms evaluation on the entire *Sales* data-mart by using the CoDe model in Fig. 1(a).

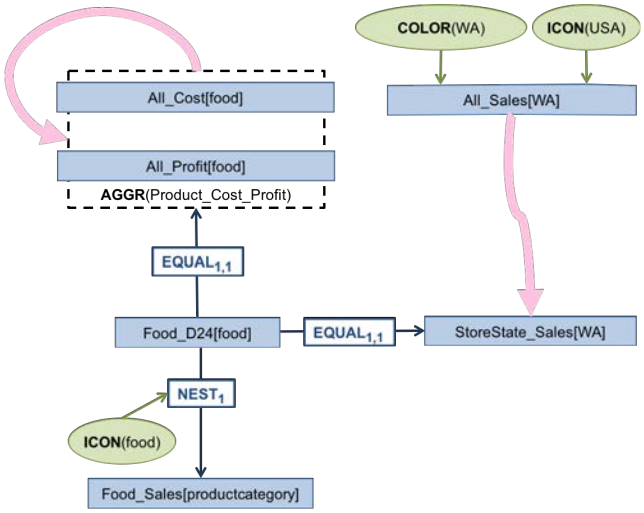
	No Mat.	**HRU$_T$**	**HRU$_S$**
Size$_p$(M)	-	327MB	177MB
Time$_t$(V, M)	95.05s	45.41s	32.28s
Time$_t$(Q, M)	38.10s	18.75s	15.16s
Profit	-	51%	60%

workload queries have to be answered. In addition, HRU_S uses less storage space than the other algorithm. In conclusion, the results assess that the algorithm HRU_S maintains good performance also on a complex CoDe model.

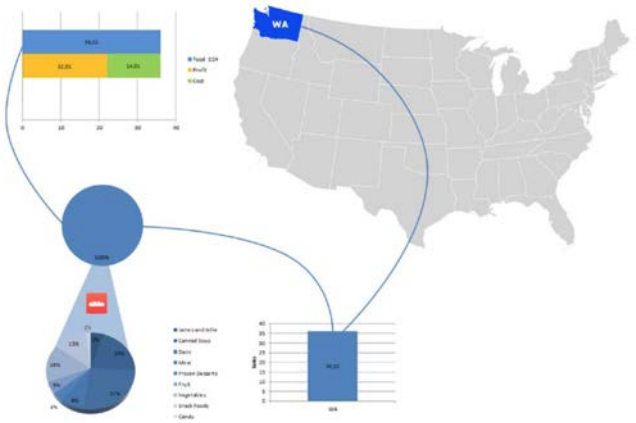
Successively, the optimization process has been applied on the CoDe model of Fig. 9(a). In particular, this model regards the cost and profit of the food category of markets in the Washington state (i.e., WA), but taking into account the fixed day (i.e., 24). The generated report is shown in Fig. 9(b). All the phases of the proposed approach has been applied starting from the CoDe model, and Fig. 10 shows the obtained lattice structure and the computed MST. In particular, the view space and the processing cost is computed for each node in according to the proposed cost model, the nodes colored in gray represent the views of the workload. For sake of space the generated vocabulary table, switchable strings and OLAP unique operation patterns have been omitted.

The HRU_T procedure (with $k = 3$) selects the views: $root$, v_2, v_{48} and v_{59}. The HRU_S procedure with storage space $s = 114$ selects the views: $root$, v_2, v_{12}, v_{13}, v_{22}, v_{32}, v_{42}, v_{46}, v_{54}. Finally, the algorithm proposed by Shukla *et al.* [7], by assuming that all aggregates have an equal probability of being queried, selects the views: $root$, v_2, v_{12}, v_{34}, and v_{43}.

Table 10 summarizes the obtained results of the three algorithms. In particular, the algorithm HRU_S reduces the total processing time to answer the workload queries and the used storage space (111MB wrt. HRU_T that uses 298MB). In this case HRU_T has a worse performance in term of processing time when all the views have to be calculated, due the. Moreover, $AvQC$ has better results in term of storage space but worst processing time wrt. HRU_S. It is worth noting that the obtained results are



(a)



(b)

Fig. 9: CoDe model for the data-mart *Sales* concerning the cost and profit of the food category in the WA state for the day 24 (a), and its graphical representation (b).

TABLE 10: Processing time and storage space of adopted algorithms applied on the lattice structure of Fig. 10.

	No Mat.	AvQC	HRU_T	HRU_S
Selected views	*root*	*root*, v_2, v_{12}, v_{34}, v_{43}	*root*, v_2, v_{48}, v_{59}	*root*, v_2, v_{12}, v_{13}, v_{22}, v_{32}, v_{42}, v_{46}, v_{54}
Size(M)	-	96MB	298MB	111MB
Time(V, M)	48.70s	21.11s	25.75s	17.16s
Time(Q, M)	24.07s	17.03s	16.89s	15.50s
Profit	-	29%	30%	36%

worse considering the ones obtained on the previous CoDe model (i.e., 36% vs. 62%), due the presence of an higher number of leaf nodes onto the MST corresponding to the workload queries wrt the MST depicted in Fig. 8. This aspect highlights how the results are affected by the lattice structure.

5 Conclusions

It this paper, we propose a process based on the CoDe modeling to detect a set of view to materialize. Through the CoDe model, the company manager expert of a specific domain designs what reports have to be visualized.

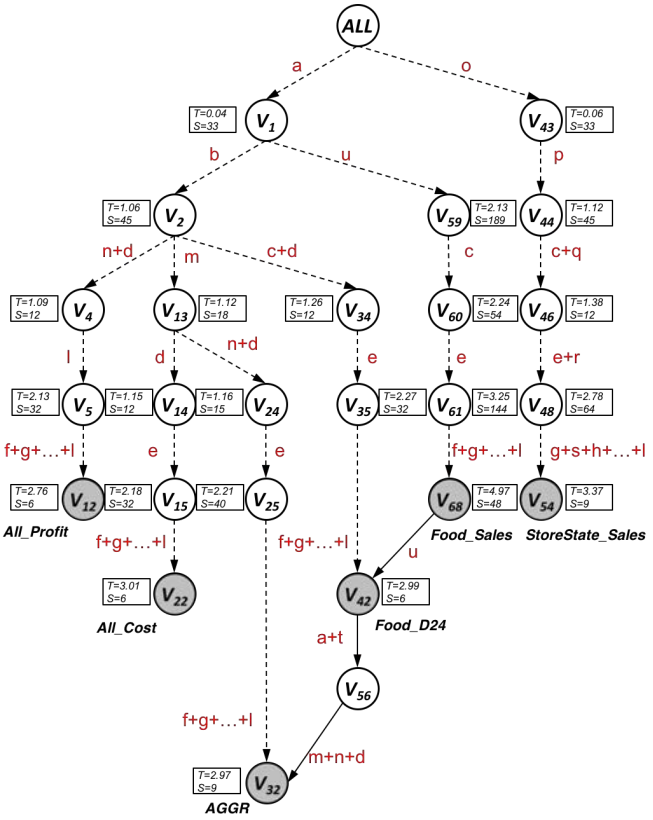


Fig. 10: The lattice structure with the OLAP operations and the corresponding MST (dashed arrows) constructed on the CoDe model in Fig. 9(a).

This information can be exploited to optimize the queries used to extract data from DW.

The proposed approach has been evaluated on a real DW obtaining an improvement on the processing time in the range of 36-62% for the algorithm HRU_S wrt. the solution which does not perform any materialization, and 7% wrt. an approach that exploits the materialized views maximizing the benefit per unit space based on their probability to be queried (i.e., $AvQC$). In the case of two consecutive executions, the algorithm HRU_S reaches an improvement of at least 98% after the second execution. This value is quite constant after other successive iterations. By considering a whole CoDe model, the results confirm the ones obtained on a sub-part of the model. In particular, the algorithms HRU_T and HRU_S reach an improvement of 51% and 60%, respectively.

To consolidate the results presented in this paper, we plan to test the proposed process on dynamic and larger DWs.

In conclusion, through the CoDe model, the company manager expert of a specific domain, designs what reports have to be visualized. This information can be exploited to optimize the queries used to extract data for that reports providing a valid decisional support. In the future, we plan to manage the incremental changes of a pre-existing CoDe model that reflect the data updates in the DW. In particular, we expect to identify the parts

involved in the changes and update the data structures used in the optimization process only by considering these parts. This further will reduce the costs needed to extract data for reports. Moreover, we shall consider to add new functionalities based on Data mining techniques, which allow to investigate the CoDe model and help the company manager to easily perform statistical analysis and to find patterns on selected data.

REFERENCES

[1] J. Widom, "Research Problems in Data Warehousing," in *Proceedings of the 4th International Conference on Information and Knowledge Management (CIKM)*, 1995, pp. 25–30.

[2] C.-S. Park, M.-H. Kim, and Y.-J. Lee, "Rewriting OLAP Queries Using Materialized Views and Dimension Hierarchies in Data Warehouses," in *Proceedings of the 17th International Conference on Data Eng. (ICDE)*, 2001, pp. 515–523.

[3] W. Lehner, W. Hummer, and L. Schlesinger, "Processing Reporting Function Views in a Data Warehouse Environment," in *Proceedings of the 18th International Conference on Data Engineering (ICDE)*, 2002, pp. 176–185.

[4] D. Habich, W. Lehner, and M. Just, "Materialized Views in the Presence of Reporting Functions," in *Proceedings of the 18th International Conference on Scientific and Statistical Database Management (SSDBM)*, 2006, pp. 159–168.

[5] S. Agrawal, S. Chaudhuri, and V. R. Narasayya, "Automated Selection of Materialized Views and Indexes in SQL Databases," in *Proceedings of the 26th International Conference on Very Large Data Bases (VLDB)*, 2000, pp. 496–505.

[6] V. Harinarayan, A. Rajaraman, and J. D. Ullman, "Implementing Data Cubes Efficiently," in *Proceedings of the International Conference on Management of Data (SIGMOD)*, 1996, pp. 205–216.

[7] A. Shukla, P. Deshpande, and J. F. Naughton, "Materialized View Selection for Multidimensional Datasets," in *Proceedings of the 24rd Intlernational Conference on Very Large Data Bases (VLDB)*, 1998, pp. 488–499.

[8] C. Zhang, X. Yao, and J. Yang, "An Evolutionary Approach to Materialized Views Selection in a Data Warehouse Environment," *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, vol. 31, no. 3, pp. 282–294, 2001.

[9] M. Risi, M. I. Sessa, M. Tucci, and G. Tortora, "CoDe Modeling of Graph Composition for Data Warehouse Report Visualization," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 3, pp. 563–576, 2014.

[10] M. Risi, M. I. Sessa, G. Tortora, and M. Tucci, "Visualizing Information in Data Warehouses Reports," in *Symposium on Advanced Database Systems (SEBD)*, 2011, pp. 246–257.

[11] C. Stolte, D. Tang, and P. Hanrahan, "Polaris: a System for Query, Analysis, and Visualization of Multidimensional Relational Databases," *IEEE Transactions on Visualization and Computer Graphics*, vol. 8, no. 1, pp. 52–65, 2002.

[12] J. Mackinlay, P. Hanrahan, and C. Stolte, "Show me: Automatic presentation for visual analysis," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 6, pp. 1137–1144, 2007.

[13] Mondrian Pentaho, *Foodmart*. [Online]. Available: http://mondrian.pentaho.com

[14] V. Indelli Pisano, M. Risi, and G. Tortora, "Exploiting CoDe Modeling for the Optimization of OLAP Queries," in *11th International Conference on Digital Information Management (ICDIM)*, 2016.

[15] J. Yang, K. Karlapalem, and Q. Li, "Algorithms for Materialized View Design in Data Warehousing Environment," in *Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB)*, 1997, pp. 136–145.

[16] H. Gupta and I. S. Mumick, "Selection of Views to Materialize in a Data Warehouse," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 1, pp. 24–43, 2005.

[17] A. Gong and W. Zhao, "Clustering-Based Dynamic Materialized View Selection Algorithm," in *Proceedings of the 5th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, vol. 5, 2008, pp. 391–395.

[18] M. Chaudhari and C. Dhote, "Dynamic Materialized View Selection Algorithm: A Clustering Approach," in *Proceedings of the International Conference on Data Engineering and Management (ICDEM)*, vol. 6411, 2012, pp. 57–66.

[19] J. Bertin, *Semiology of Graphics : Diagrams, Networks, Maps*. University of Wisconsin Press Madison, Wis, 1983.

[20] S. Chaudhuri and U. Dayal, "An Overview of Data Warehousing and OLAP Technology," *SIGMOD Record*, vol. 26, no. 1, pp. 65–74, 1997.

[21] A. Valmari, "The State Explosion Problem," in *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, the Volumes Are Based on the Advanced Course on Petri Nets*. Springer-Verlag, 1998, pp. 429–528.

[22] H. Karloff and M. Mihail, "On the Complexity of the View-selection Problem," in *Proceedings of the 18th Symposium on Principles of Database Systems (PODS)*, 1999, pp. 167–173.

[23] I. Mami and Z. Bellahsene, "A Survey of View Selection Methods," *SIGMOD Record*, vol. 41, no. 1, pp. 20–29, 2012.

[24] G. Chan, Q. Li, and L. Feng, "Optimized Design of Materialized Views in a Real-Life Data Warehousing Environment," *International Journal of Information Technology*, vol. 7, no. 1, pp. 30–54, 2001.

[25] E. Baralis, S. Paraboschi, and E. Teniente, "Materialized Views Selection in a Multidimensional Database," in *Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB)*, 1997, pp. 156–165.

[26] B. Shah, V. Ramachandran, and K. Ramachandran, "A Hybrid Approach for Data Warehouse View Selection," *International Journal of Data Warehousing and Mining*, vol. 2, no. 2, pp. 1–37, 2006.

[27] Meteorite Saiku, *Open Source Analysis Suite*. [Online]. Available: http://analytical-labs.com