Hierarchical Reorganization of Dimensions in OLAP Visualizations

Sébastien Lafon, Fatma Bouali, Christiane Guinot, and Gilles Venturini

Abstract—In this paper, we propose a new method for the visual reorganization of online analytical processing (OLAP) cubes that aims at improving their visualization. Our method addresses dimensions with hierarchically organized members. It uses a genetic algorithm that reorganizes k-ary trees. Genetic operators perform permutations of subtrees to optimize a visual homogeneity function. We propose several ways to reorganize an OLAP cube depending on which set of members is selected for the reorganization: all of the members, only the displayed members, or the members at a given level (level by level approach). The results that are evaluated by using optimization criteria show that our algorithm has a reliable performance even when it is limited to 1 minute runs. Our algorithm was integrated in an interactive 3D interface for OLAP. A user study was conducted to evaluate our approach with users. The results highlight the usefulness of reorganization in two OLAP tasks.

Index Terms—Dimension reorganization, visual OLAP, interactive knowledge discovery

1 INTRODUCTION

On-line analytical processing (OLAP) [1] [2] can be viewed as a user-centered approach for analyzing multidimensional data from databases (please refer to [3] for an introduction to OLAP). In OLAP, the user explores a data cube, i.e., a multidimensional representation of data (see Section 2.1 for more details), in a visual and interactive way. Many visual interfaces were proposed to explore such data cubes (see, e.g., a survey of 2D visualizations in [4] and of 3D visualizations in [5]). In such visualizations, reorganizing the values of the dimensions is often necessary to facilitate the knowledge discovery process and to place next to each other data that are similar. In Fig. 1, we show an example of a cube before and after reorganization. The reorganization helps to detect similarities or dissimilarities in the data; such discovered information can be groups of similar data, relations between such groups, or outliers. This reorganization can be manual, with the use of specific OLAP operators. However, it was shown that manual reorganization of visualizations is tedious for users [6]. As a result, researchers developed automatic approaches, but only a few of them were tested with OLAP (see Section 2.3). In addition, these approaches cannot address tree-structured dimensions, which are common in OLAP.

Thus, the main objectives of this paper are to 1) develop visual reorganization algorithms that can reorder tree-structured dimensions, and 2) integrate and test these algorithms in an OLAP user interface.

The remainder of this paper is organized as follows: In Section 2, we introduce basic terms and definitions about OLAP and the reorganization of data cubes, and we present the state of the art in visual reorganization with a focus on the reorganization of trees and OLAP cubes. In Section 3, we present the main principles of our genetic algorithm (GA), which we use to optimize trees. In Section 4, we suggest the use of level by level reordering algorithms. In Section 5, we present the experimental results in terms of the optimization of the performances and the visual effects, and in Section 6, we detail a user study. In Section 7, we present the conclusions and perspectives that can be derived from this work.

2 STATE OF THE ART IN THE REORGANIZATION OF OLAP VISUALIZATIONS

2.1 Terms and Definitions for the OLAP Cube’s Visual Reorganization

We provide more details in this section about the terms that are used to describe an OLAP data cube and the reorganization of the dimensions in OLAP. A data cube is built on a core data structure called a fact table. A fact represents an elementary chunk of information that is described with a set of measures and a set of dimensions. For example, if one considers sales about products, a fact could be “a beer was sold in Los Angeles at price x.” The price (subsequently denoted “Sales”) is called a measure. This value is specific because it can be aggregated: one could sum the sales of all of the beers, or the sales of all of the stores. The dimensions can be “Product” and “Store.” The values of a dimension are called the members. The members are organized into sets called levels. Such members may have a “flat” representation, that is: they all belong to the same level. This corresponds to the usual dimension’s representation of many data exploration methods. However, in OLAP, the members of a dimension are very often organized with hierarchical levels. For example, the “Store” dimension can be organized into three levels “Country=(USA,…),” “State=(California,…),” and...
explore data at different levels of description (e.g., location) is fundamental to OLAP because it allows users to find products with similar sales).

Each dimension is composed of tree-structured members. In the following, we will represent the hierarchical organization of the levels with a tree ($T_1$ and $T_2$ in the figure). Each node of the tree represents one member. The matrix with gray levels represents the values of the measure (e.g., “Sales”). The aim of the reorganization is to rearrange the members of each dimension to reveal the underlying structure of the measure and of the members (as shown from the left to the right in the figure, where the trees were reorganized to place cells with similar values of the measure nearby each other). In this example, the reorganization helps to discover knowledge such as “similar products are sold in Oregon and Washington state,” or “food is sold in three states only.” Without such reorganization, discovering such knowledge can be possible, but it requires more attention and effort from the user. With this example, we also underline that accounting for the levels in the reorganization is important. If the members are reorganized without any hierarchical constraints, then products from the “FOOD” subtree could be intermixed with products from the “DRINK” subtree. This outcome could be interesting if the user is looking for correlations between the sales of these two subtrees (i.e., finding drinks and foods with similar sales). However, if the user is looking for groups of similar products within each subtree, then the tree structure must be accounted for (i.e., finding groups of drinks with similar sales). In addition, the hierarchical reorganization can account for the adjacency of the subtrees; in the reorganized “Store” dimension, the states “Oregon” and “Washington” are placed far away from the states “British Columbia” and “Ontario” because their sales are very different. The sales of “California” are placed between the others because they are more similar to each of the other states. Thus, the hierarchical reorganization can create groups within subtrees and might also find a global organization of the trees.

In OLAP, several operators are related to the cube’s reorganization, like, for example, the “Switch” operator, which permutes two adjacent members along a given dimension. However, the manual reorganization of an OLAP cube could be too difficult a task for the user because several dimensions with many members are displayed at the same time. Therefore, as it will be shown in Section 2.3, several algorithms were developed to reorganize the visual representations of data cubes.

### 2.2 Visual Reorganization: General Case

Reordering the elements of a visualization is the subject of many studies, which we briefly summarize here. First, Ankerst et al. [7] showed that this problem is NP-complete. This explains why the manual approaches might be tedious for users. Several cases of reorganization problems can be distinguished. The visual elements can be reorganized in 1D, 2D, or 3D, and the elements can also be structured in a hierarchy.

The problem of the 1D visual reorganization concerns visualizations in which the visual elements must be rearranged in a linear way. Studies were devoted to the reordering of the axes of the parallel coordinates visualization [23]; in [7], a permutation of the axes must be found. In
a permutation of the axes and a spacing between these axes is optimized. In Radviz [24], the order of the dimensions is also rearranged [9]. This 1D reorganization problem can be related to the traveling salesman problem [7], [10] because the visual elements to be reorganized represent cities, and the similarities between the elements represent the distances between the cities. The problem, thus, is to find a permutation of the visual elements that minimizes the sum of the distances between the adjacent elements. It is also important to notice that the 2D (or 3D) reorganization problem can often be formulated as two (respectively, three) 1D reorganization problems in which the dimensions are optimized independently from each other. For instance, in matrices reorganization, most of the approaches optimize the rows and the columns independently.

We consider now the 2D reorganization problem, in which two orderings must be found. This problem concerns mainly the matrix reorganization problem, which was the subject of many studies (see the following surveys [25], [26], [27]). Interactive approaches are possible, such as in the pioneering work of Bertin [12]. However, it was shown in several user studies (see an example in [6]) that users might prefer to use an automatic approach. A study conducted by McCormick et al. [11] presents the bond energy algorithm (BEA). The BEA uses a heuristic to separately reorganize the rows and columns of a matrix. It starts by randomly choosing a row (or a column) and inserts the other rows one by one at a location that maximizes a “measure of effectiveness.” In Section 4, we will propose a hierarchical version of this algorithm. Other methods for the 2D reorganization problem should be mentioned, like, for instance, [13] and [10]. As far as the 3D reorganization of visual elements is concerned, the only references that we found in the literature address the 3D layout of graphs and trees. All these approaches do not deal with hierarchies and cannot be applied to the reorganization of an OLAP cube.

Concerning the problem of the visual reorganization of tree-structured data, as mentioned in Section 1, early research concerned dendrograms. Dendrograms are binary classification trees that are generated by agglomerative hierarchical clustering algorithms (AHC), such as [28]. Several studies in this context can be mentioned, such as [14], [15] (in which an optimal algorithm is proposed with a complexity of $O(n^4)$), and [16] (in which a simulated annealing algorithm is used). However, these approaches are limited to binary trees, while tree-structured dimensions in OLAP are very often k-ary trees. In [17], a generalization of [15] is proposed. It can reorganize k-ary trees. This algorithm is based on the following idea: the k-ary tree is turned into a binary tree by adding fictive nodes. Its time complexity is high ($O(4^kn^{3/2})$ in the general case), and this algorithm is, in fact, devoted to trees for which $k$ is small (by construction).

Another attempt to visually reorganize hierarchies is presented in [18]. In this study, data to be displayed is not initially hierarchical. A hierarchy of dimensions is built according to a similarity measure between the dimensions. The authors mention five ways to reorganize such a hierarchy: 1) no reordering (one assumes that the method that builds the hierarchy places similar dimensions next to each other in the tree), 2) manual editing, 3a) an exhaustive search that performs all possible permutations, 3b) a hill-climbing search method that performs random permutations of subtrees, and 3c) the use of a heuristic to sort the subtrees (here, weights that are learned by a principal component analysis are used to order the dimensions). However, in their paper, the authors did not evaluate these automatic methods in terms of their optimization performances (i.e., improvements quantified by a cost function) or in terms of user performances (with a user study). Furthermore, this approach is not related to OLAP.

### 2.3 Visual Reorganization of OLAP Cubes

Before we describe the existing approaches for the reorganization of OLAP cubes, we mention that reorganization methods exist for OLAP, but are outside the context of
visualization. These approaches (e.g., [29]) aim at compressing a data cube but not at improving its visual aspects.

As far as visual reorganization is concerned, we can mention the study described in [19], in which the authors permute the rows and columns of an OLAP cube to obtain a Robinson matrix; however, this approach does not address hierarchical levels. The study in [20] describes a new OLAP operator that uses AHC to group together the members of a dimension. This approach can lead to interesting reorganizations, but it does not address hierarchical levels. In [21], another method based on multiple correspondence analysis is used to reorder the members, but again it does not address the hierarchies. In our initial work, we studied a simple GA to reorganize an OLAP cube [22]; however, this approach was not very efficient (a GA run was too long), and it did not address the hierarchies.

We next summarize the required properties of dimension reorganization method for visual OLAP (see Table 1 for an overview of all of the references cited in this paper). The method should address tree-structured dimensions with no specific assumptions about the shapes of the trees. It should be integrated with an interactive user interface, which means that the reorganization should be considered to be an OLAP operator that users can trigger at any time. In addition, the response time of the reorganization method should be as short as possible.

3 PRINCIPLES OF A GA FOR THE VISUAL REORGANIZATION OF OLAP HIERARCHIES

3.1 Motivations for Using a Genetic Algorithm

GAs [30] are used as stochastic search procedures and have been successful in many domains. GAs search for an optimal solution that maximizes a fitness function. GAs use a population of individuals that evolve with a selection procedure and with genetic operators. Each individual represents a possible solution to the problem. From an initial population (which, in general, is randomly generated), GAs generate a new population as follows: Individuals are evaluated with the fitness function. The best individuals are selected for reproduction. Offspring are generated from the parents with a crossover operator and with a mutation operator. These operators modify the genes of the individuals, i.e., elementary parts of the solutions. The crossover operator combines the genes of two parents to produce better individuals. The mutation operator introduces a random noise in the individuals. After several generations, GAs can find individuals with high fitness values.

For the problem that we address here, GAs have interesting properties that serve well the two goals of this paper: GAs have been especially adapted for working with trees. The so-called genetic programming (GP) approach of Koza [31] provides an effective framework for optimizing trees and was applied to many different domains with success. In addition, as shown in the next section, we must perform permutations of nodes in our OLAP hierarchies, and many studies on GAs have proposed genetic representations and operators for permutation-based search spaces [32]. GAs can be slower than other methods; however, we believe that they can fit with interaction constraints. Specifically, GAs can be started or stopped at any time, and can, therefore, provide a (partial) result even after a short time. In this way, we believe that GAs can be well integrated in an OLAP interactive interface in which the user can start/stop the optimization at will. Finally, GAs perform a global optimization and can escape from local minima more easily than search methods that are based on a single point.

Applying a GA to a tree reorganization problem requires defining a genetic representation for the solutions (i.e., an encoding for the genes), a fitness function of these solutions, genetic operators and a global design of the evolutionary algorithm.

3.2 Representation and Fitness of Individuals

Let us consider an OLAP cube \( C \) that has three dimensions \( \{D_1, D_2, D_3\} \), in which some (or all) of the dimensions are hierarchically organized. Let \( \{T_1, T_2, T_3\} \) denote the trees that correspond to each dimension of the cube. Each individual in the population represents a possible reorganization of \( C \). For a given tree \( T_i \), we define a set of permutations \( S_i \), for which each permutation of \( S_i \) is associated with each nonleaf node \( N \) of \( T_i \). Such a permutation is local to \( N \) and directly influences the order of its child nodes. Thus, an individual in our GA is represented as three sets of permutations \( \{S_1, S_2, S_3\} \), one for each dimension.

Fig. 3 illustrates this genetic encoding for a single tree \( T_1 \) (“Store” dimension). For each nonleaf node, a permutation is defined. For example, the permutation \( S_{1.1} \) corresponds to the root node of the tree (the “All” level). It contains three elements (one for each of the child nodes “Canada,” “USA,” and “Mexico”). The global encoding is obtained by the concatenation of all of the permutations, in a breadth-first order.

The fitness of a cube \( C \) is defined by computing its “visual homogeneity” \( H(C) \), as was initially presented in [20] and [21]. This value is obtained by locally computing the similarity between each cell and its neighboring cells and by summing up all of these similarities. More precisely, the similarity between a cell and its neighborhood is computed as follows:

<table>
<thead>
<tr>
<th>References</th>
<th>Reorganized structure</th>
<th>Hierarchical constraints</th>
<th>Used for OLAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>[7] [8] [9] [10]</td>
<td>Axes of visualization</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>[14] [15] [16]</td>
<td>Dendrograms (binary trees)</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>[17] [18]</td>
<td>k-ary Trees</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>[19] [20] [21] [22]</td>
<td>OLAP 2D cubes</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>

As far as we know, no hierarchical reorganization algorithm was applied to OLAP visualizations.
where \( \text{Cell} \) is the considered cell, \( m(\text{Cell}) \) is the measure's value for this cell, and \( \vartheta(\text{Cell}) \) is the set of the cells that are located next to \( \text{Cell} \) (we use a Moore neighborhood, which represents here up to 26 cells in 3D). Finally, the fitness of an individual equals:

\[
H(C) = \sum_{\text{Cell} \in \vartheta(\text{Cell})} \frac{s(\text{Cell})}{\max(C) - \min(C)},
\]

where \( \max(C) \) and \( \min(C) \) represent the maximum and minimum values of the measure of \( C \). This fitness function favors cubes for which neighboring cells contain similar values; it is a cost function that must be minimized. If two measures \( M_1 \) and \( M_2 \) are defined for \( C \), then the fitness of an individual is \( H_1(C) + H_2(C) \), where \( H_1 \) and \( H_2 \) are the values of \( H \) for \( M_1 \) and \( M_2 \), respectively.

### 3.3 Genetic Operators

To generate an initial population, we consider the cube \( C \) that is currently displayed when the user triggers the reorganization. According to the user's choice, one individual that represents the initial cube \( C \) is created in the population. Other individuals of the initial population are randomly generated by performing random permutations on all of the nodes of \( T_1, T_2, \) and \( T_3 \). In this way, our method can account for any rearrangement that was manually performed by the user. The global structure of the GA is such that this individual will survive in the population as long as it is not the worst individual (see Section 3.4).

The mutation operator considers a parent individual \( P \) and selects, on each dimension of \( P \), a permutation (subtree) to be rearranged. More precisely (see Fig. 4), for a given tree \( T \), this operator randomly chooses an internal node of \( T \) and modifies the corresponding permutation with a probability of \( p_{\text{mut}} \). Two mutation algorithms were implemented and they are inspired by the work performed on the genetic resolution of the traveling salesman problem [32]: the “City-Swap” chooses two nodes within a permutation and exchanges their location in the sequence. The “2-opt” selects two nodes within a permutation and inverts the subsequence that is defined by these two nodes.

The crossover operator uses the following main principles (see Fig. 5): for two parents \( P_1 \) and \( P_2 \), it selects, for each dimension, a subtree to be recombined. For two corresponding subtrees to be recombined from the parents, a crossover is performed on the corresponding two permutations, and the resulting combined permutation replaces the parent permutation in \( P_1 \) (this individual
becomes the output of the crossover operator). We used two crossover operators that combine pairs of corresponding permutations (see [32]): the uniform crossover (UX) which, for a couple of corresponding permutations (one from $P_1$ and the other from $P_2$), selects the current node in the offspring either from $P_1$ or $P_2$. If a node is already present in the offspring, then this operator selects the next node of the parent that is not in the offspring. The other crossover operator is the ordered crossover (OX, see [33]), which, for each pair of corresponding permutations, randomly defines two cutting points. All of the nodes between these cutting points are exchanged. If a node is present twice in an offspring, then the second occurrence of this node is replaced by the next missing node in the offspring.

### 3.4 Main Algorithm

We used a binary tournament selection operator to avoid any scaling problems in the fitness function. With this operator, to select one individual, two individuals are randomly chosen, and their fitnesses are compared. The best individual is selected. For inserting new individuals in the population, we used Whitley’s “steady state” method, which replaces the worst individual in the population by the newly created offspring [34]. With this method, one individual is generated at each generation, and if this individual has a better fitness than the worst individual in the population, then it replaces the worst individual. The main algorithm that results from these principles is as follows:

1. Generate an initial population (possibly including the cube that was manually reorganized by the user).
2. Generate one individual (with the “steady state” method):
   a. Select two parents, $P_1$ and $P_2$, with a binary tournament.
   b. $I \leftarrow \text{Crossover}(P_1, P_2)$ with probability $p_{\text{cross}}$ else $I \leftarrow P_1$.
   c. $I \leftarrow \text{Mutation}(I)$.
   d. Compute $I$’s fitness and possibly insert $I$ into the population.
3. If the maximum number of generations is reached, then stop and output the best individual, else go to 2.

### 4 Several Types of Visual Reorganization for OLAP

One might consider that there are at least three different ways of applying a reorganization algorithm to an OLAP cube (see Fig. 6). The first possibility is to reorganize the cube as displayed by the user. This version is the default version of our method, which was described in the previous section. The advantage of this method is its computational cost because it works on the displayed cube and not on the complete hierarchies. For example, if a large hierarchy is displayed at its first level only, then only that level will be reorganized. A second characteristic of this method is that it uses aggregated values of the measure when a hierarchy is not completely developed.

The second possibility is to reorganize the whole cube, regardless of the part that is currently displayed. In this case, the reorganization algorithm is applied to completely developed hierarchies, and no aggregated measures are used (only the values that are present in the leaves of the trees are used). This version might have a higher computational cost because it addresses complete trees. However, it can be applied offline before the dimensions are displayed, and no reorganization is necessary during the exploration of the cube (unless the user changes the displayed dimensions).

The third possibility is to reorganize the hierarchies with a level by level approach, using an aggregation of measure values. In this case, the reorganization algorithm is run several times, one for each level of the hierarchies. This approach can be related to the method that is presented in [18], in which hierarchies are reorganized node by node with aggregated values. In the first run, it reorganizes the first level of each hierarchy, with aggregated values. The second levels are reorganized during the second run, and so on. Once a level is reorganized, it is not modified anymore by the succeeding runs of the algorithm on lower levels. This version of the reorganization might be interesting because it considers the aggregated values (which can often be computed in OLAP). Selecting this version depends on the data and the expert preferences. For example, if the “Country” level is considered, then the countries with similar measures will be placed next to each other. Subsequently, when the sublevel “City” is reorganized, the order of the upper level “Country” will not be modified.

We propose two “level by level” hierarchical reorganization algorithms that, as far as we know, have not been studied yet for OLAP. The first algorithm uses our genetic approach and this algorithm is denoted by “Level-GA” in the following. We use the same GA as presented in the previous section, with the exception that this algorithm is applied to each level. The second algorithm that we propose is an adaptation of the BEA [11]. The BEA is not a hierarchical algorithm; however, this “level by level” approach does not require a hierarchical reorganization algorithm because each level is treated individually as a linear structure. We denote this second algorithm “Level-BEA.” The Level-BEA works as follows: It optimizes the dimensions one by one (the BEA cannot optimize dimensions together but optimizes them separately). For a given dimension, the Level-BEA reorders the
first level. The order of these nodes is then fixed. Then, the Level-BEA reorganizes the second level of the dimension. However, instead of considering any possible location at which to insert the members of the considered level, it considers only the locations that respect the constraints that are imposed by the ordering of the upper levels (the result must be a valid tree).

These level by level approaches can be applied only to reorganization problems in which values that are present on the leaves can be aggregated in the upper levels, which is the case for OLAP.

5 EXPERIMENTAL STUDY

5.1 Tested Data

Several experiments were conducted. First, we attempted to determine the best parameters of the GA. In general, this task is not easy and is highly domain dependent. Then we performed a comparative evaluation with different algorithms on the test cases that were not used for finding the best parameters. For these tests (see Table 2), we used the data cubes that are distributed with the Mondrian OLAP server (mondrian.pentaho.com, which is the OLAP server that was used in our implementation). The database is named “MondrianFoodMart,” and many data cubes can be defined from it. We selected five cubes to address hierarchies. As a consequence, the possible values present on the leaves can be aggregated in the upper levels, which is the case for OLAP.

<table>
<thead>
<tr>
<th>Cube</th>
<th>Dim(Level)</th>
<th>Measure(s)</th>
<th>Nb of Cells</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cube 1 (“HR”)</td>
<td>Time(3), Store(4), Pay Type(1)</td>
<td>“Org Salary”</td>
<td>24 × 25 × 2 = 1200</td>
</tr>
<tr>
<td>Cube 2 (“Sales”)</td>
<td>Promotion media(1), Store-size(1), Promotions(1)</td>
<td>“Unit sales”</td>
<td>14 × 21 × 51 = 17136</td>
</tr>
<tr>
<td>Cube 3 (“Sales”)</td>
<td>Store(2), Promotions(1), Product(2)</td>
<td>“Unit sales”</td>
<td>10 × 51 × 23 = 11730</td>
</tr>
<tr>
<td>Cube 4 (“Sales”)</td>
<td>Store Size in SQRT(1), Time(2), Product(2)</td>
<td>“Unit sales”, “Customer count”</td>
<td>21 × 8 × 23 = 3864</td>
</tr>
<tr>
<td>Cube 5 (“Sales”)</td>
<td>Store Size in SQRT(1), Time(3), Product(2)</td>
<td>“Unit sales”</td>
<td>21 × 24 × 23 = 11592</td>
</tr>
</tbody>
</table>

We used the “HR” and “Sales” cubes as a basis for defining five 3D cubes. “Dim(Level)” means that for dimension “Dim” the displayed level is “Level.” Measures (“Org Salary,” “Unit sales,” or “Customer count”) were selected among those that were available.

5.3 Comparative Study

We compared our GA with several other methods. We remind the reader that, in the literature, there is no existing heuristic but applies it level by level on the trees. In the

We performed a comparative evaluation with different algorithms on the test cases that were not used for finding the best parameters. For these tests (see Table 2), we used the data cubes that are distributed with the Mondrian OLAP server (mondrian.pentaho.com, which is the OLAP server that was used in our implementation). The database is named “MondrianFoodMart,” and many data cubes can be defined from it. We selected five cubes to address hierarchies. As a consequence, the possible values present on the leaves can be aggregated in the upper levels, which is the case for OLAP.

<table>
<thead>
<tr>
<th>Cube</th>
<th>Dim(Level)</th>
<th>Measure(s)</th>
<th>Nb of Cells</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cube 1 (“HR”)</td>
<td>Time(3), Store(4), Pay Type(1)</td>
<td>“Org Salary”</td>
<td>24 × 25 × 2 = 1200</td>
</tr>
<tr>
<td>Cube 2 (“Sales”)</td>
<td>Promotion media(1), Store-size(1), Promotions(1)</td>
<td>“Unit sales”</td>
<td>14 × 21 × 51 = 17136</td>
</tr>
<tr>
<td>Cube 3 (“Sales”)</td>
<td>Store(2), Promotions(1), Product(2)</td>
<td>“Unit sales”</td>
<td>10 × 51 × 23 = 11730</td>
</tr>
<tr>
<td>Cube 4 (“Sales”)</td>
<td>Store Size in SQRT(1), Time(2), Product(2)</td>
<td>“Unit sales”, “Customer count”</td>
<td>21 × 8 × 23 = 3864</td>
</tr>
<tr>
<td>Cube 5 (“Sales”)</td>
<td>Store Size in SQRT(1), Time(3), Product(2)</td>
<td>“Unit sales”</td>
<td>21 × 24 × 23 = 11592</td>
</tr>
</tbody>
</table>

We used the “HR” and “Sales” cubes as a basis for defining five 3D cubes. “Dim(Level)” means that for dimension “Dim” the displayed level is “Level.” Measures (“Org Salary,” “Unit sales,” or “Customer count”) were selected among those that were available.

We performed a comparative evaluation with different algorithms on the test cases that were not used for finding the best parameters. For these tests (see Table 2), we used the data cubes that are distributed with the Mondrian OLAP server (mondrian.pentaho.com, which is the OLAP server that was used in our implementation). The database is named “MondrianFoodMart,” and many data cubes can be defined from it. We selected five cubes to address hierarchies. As a consequence, the possible values present on the leaves can be aggregated in the upper levels, which is the case for OLAP.

<table>
<thead>
<tr>
<th>Cube</th>
<th>Dim(Level)</th>
<th>Measure(s)</th>
<th>Nb of Cells</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cube 1 (“HR”)</td>
<td>Time(3), Store(4), Pay Type(1)</td>
<td>“Org Salary”</td>
<td>24 × 25 × 2 = 1200</td>
</tr>
<tr>
<td>Cube 2 (“Sales”)</td>
<td>Promotion media(1), Store-size(1), Promotions(1)</td>
<td>“Unit sales”</td>
<td>14 × 21 × 51 = 17136</td>
</tr>
<tr>
<td>Cube 3 (“Sales”)</td>
<td>Store(2), Promotions(1), Product(2)</td>
<td>“Unit sales”</td>
<td>10 × 51 × 23 = 11730</td>
</tr>
<tr>
<td>Cube 4 (“Sales”)</td>
<td>Store Size in SQRT(1), Time(2), Product(2)</td>
<td>“Unit sales”, “Customer count”</td>
<td>21 × 8 × 23 = 3864</td>
</tr>
<tr>
<td>Cube 5 (“Sales”)</td>
<td>Store Size in SQRT(1), Time(3), Product(2)</td>
<td>“Unit sales”</td>
<td>21 × 24 × 23 = 11592</td>
</tr>
</tbody>
</table>

We used the “HR” and “Sales” cubes as a basis for defining five 3D cubes. “Dim(Level)” means that for dimension “Dim” the displayed level is “Level.” Measures (“Org Salary,” “Unit sales,” or “Customer count”) were selected among those that were available.
same way, we tested our GA with the same level by level approach (denoted by the Level-GA). Finally, with the aim of minimizing the running times while maintaining good performance, we hybridized the Level-BEA and the GA approaches. Because the BEA (and the Level-BEA) starts from an “empty” reorganization list, one cannot use the Level-BEA to directly refine a solution that is produced by the GA. The opposite is, however, possible: The Level-BEA can be used to propose a good initial solution to the GA. The resulting algorithm is denoted by the Level-BEA+GA. The behavior of the Level-BEA (and the Level-BEA+GA) is such that it optimizes the dimensions separately and not together (i.e., the reorganization of dimension X does not influence the reorganization of Y and Z). As a result, it does not account for the dependences that can exist between dimensions. These are the reasons why this approach is not as efficient as the other approaches. As a consequence, we can also explain why the Level-BEA+GA hybridization does not always obtain the best performance: The initial starting point provided by the Level-BEA is not as helpful to the GA as we expected. However, the running times are lowered. Finally, the Level-GA is not as efficient as the GA because using the aggregated values does not appear to locally guide the reordering algorithm to the best overall solutions. Once a choice is made at a given level, it constrains the optimization of the lower levels in a way that prevents it from finding a globally better solution. Thus, the conclusions that can be derived from these tests are the following. The GA has the most reliable performance compared with the other algorithms, which can obtain good performance for some cubes but can also obtain poor performances for other cubes. However, the GA is also among the slowest methods, unlike the hybrid or heuristic approaches. As a result, we believe that hybridization might be an interesting perspective to develop, especially if we can find an efficient way to combine the GA and the BEA (e.g., the BEA could be considered to be a heuristic-based mutation operator).

### 5.4 Visual Results

In this section, we present visual results and whether they confirm or not the quantitative results of the previous section. In Fig. 7 (and in Fig. 1 for cube 5), we present the tested cubes as they are initially displayed (no reorganization) and how these cubes are modified with our GA. These visualizations confirm the visual improvement that is performed by the GA optimization.

In Fig. 8, we present the results that were obtained by each method on a given cube (cube 4). A first comment is that the visual improvement (from the RANDOM method to the other methods) can be correlated with the quantitative improvement measured in Table 3 (see the “Cube 4” column in the table). Then, the different methods can be compared. The RANDOM reorganization is clearly far from the optimal, and its results are very different as compared with the other methods. The HC and GA obtain a similar result, in which three clusters appear (drink, food, and nonconsumable). However, when we zoom on the details (such as the corner at the top of the visualizations), the GA performs a more efficient optimization than HC, and it places members next to each other that are more similar. The behavior of the Level-BEA (and the Level-BEA+GA) is very different, also, from the other methods: this approach tends to create a large cluster at the center of the visualization. This result arises because of the “effectiveness” function, which is used in the BEA. This function is

**TABLE 3**

Results Obtained by the Different Algorithms and for the Five Cubes Mentioned in Table 2, Showing an Average over 10 Trials

<table>
<thead>
<tr>
<th>Methods</th>
<th>Cube 1</th>
<th>Cube 2</th>
<th>Cube 3</th>
<th>Cube 4</th>
<th>Cube 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>RANDOM</td>
<td>1831.2 ± 9.9</td>
<td>1973 ± 2.8</td>
<td>4726 ± 6.6</td>
<td>7097.4 ± 181.3</td>
<td>6510 ± 151.5</td>
</tr>
<tr>
<td></td>
<td>17.0 ± 1.3</td>
<td>53.9 ± 0.6</td>
<td>58.4 ± 1.0</td>
<td>36.4 ± 0.4</td>
<td>68.4 ± 1.5</td>
</tr>
<tr>
<td></td>
<td>16.2 ± 0.5</td>
<td>54.1 ± 1.1</td>
<td>58.7 ± 0.3</td>
<td>36.8 ± 0.15</td>
<td>68.1 ± 1.0</td>
</tr>
<tr>
<td></td>
<td>1671.0 ± 55.4</td>
<td>193.9 ± 30.2</td>
<td>4659 ± 101.8</td>
<td>4723 ± 116.9</td>
<td>3657 ± 193.9</td>
</tr>
<tr>
<td></td>
<td>1754.9 ± 10.4</td>
<td>184.9 ± 0.6</td>
<td>4151 ± 11.4</td>
<td>4078 ± 206.6</td>
<td>3456 ± 159.1</td>
</tr>
<tr>
<td></td>
<td>17.4 ± 0.3</td>
<td>54.4 ± 0.7</td>
<td>59.4 ± 0.3</td>
<td>37.5 ± 0.3</td>
<td>69.7 ± 0.4</td>
</tr>
<tr>
<td></td>
<td>1929.3 ± 46.2</td>
<td>358.5 ± 12.9</td>
<td>714.2 ± 67.3</td>
<td>4530 ± 174.6</td>
<td>4420 ± 364.1</td>
</tr>
<tr>
<td></td>
<td>0.2 ± 0.0</td>
<td>1.4 ± 0.1</td>
<td>3.9 ± 0.1</td>
<td>1.1 ± 0.1</td>
<td>1.9 ± 0.3</td>
</tr>
<tr>
<td></td>
<td>1616.9 ± 1.8</td>
<td>184.6 ± 1.1</td>
<td>408.4 ± 4.4</td>
<td>4164.1 ± 55.9</td>
<td>3507 ± 85.8</td>
</tr>
<tr>
<td></td>
<td>32.9 ± 0.9</td>
<td>55.9 ± 1.0</td>
<td>46.2 ± 0.4</td>
<td>36.5 ± 0.25</td>
<td>73.8 ± 10.5</td>
</tr>
<tr>
<td></td>
<td>9.2 ± 0.3</td>
<td>29.9 ± 1.8</td>
<td>36.1 ± 1.3</td>
<td>20.5 ± 0.23</td>
<td>38.5 ± 11.2</td>
</tr>
<tr>
<td>Level-BEA</td>
<td>1761.6 ± 13.5</td>
<td>1864 ± 0.9</td>
<td>4210 ± 19.1</td>
<td>4260 ± 413.0</td>
<td>4039.5 ± 387.6</td>
</tr>
<tr>
<td></td>
<td>9.2 ± 0.3</td>
<td>29.9 ± 1.8</td>
<td>36.1 ± 1.3</td>
<td>20.5 ± 0.23</td>
<td>38.5 ± 11.2</td>
</tr>
</tbody>
</table>

For a given method, the first line in the table indicates the obtained cost value and the second line gives the running time (mean ± standard deviation).
more sensitive to the presence of a value than to its absence. Thus, the Level-BEA attempts to cluster the largest values together, regardless of the consequences on the other parts of the reorganization. In addition, the optimization of the dimensions is performed separately, and it fails to find dependences between the dimensions. The Level-GA illustrates the constraints that are created when optimizing the cube level by level. Two groups are defined (the upper left and lower right) when optimizing the first levels. Then, the algorithm has no way to change the organization of the upper levels when it optimizes the lower levels. Thus, it is trapped in a solution that cannot be the optimum.

6 USER STUDY

6.1 Integration with an Interactive Interface

OLAP is a user-centered approach that is tightly coupled to the visual and interactive exploration of data. Therefore, it
is crucial to integrate our reorganization approaches into a user-friendly interface, especially if one wants to evaluate its effectiveness with respect to users. This is the second aim of this paper. In our research on OLAP, we developed an interactive interface (called VR4OLAP). We pursued the work of [35], in which an immersive interface was proposed for OLAP. In our visualization, the OLAP cube is represented with a 3D cube, and each cell has a 3D visual element for which the size depends on the measure value (an example is given in Fig. 1). Three hierarchical dimensions and up to two measures might be represented in this way.\(^1\) In addition, many interactions are possible: several OLAP operators were integrated in this visualization using clickable 3D objects that are located next to the data that they work on. When clicking on such an object, the user triggers the corresponding operator on the data that is located nearby. The interested reader can refer to [5] for further details.

With regard to the reorganization, the user can perform a manual optimization by triggering the Switch operator, for example. However, he might use an automatic approach and trigger one algorithm that is selected among those studied in this paper. The selected algorithm is run in an incremental way, with steps of at most 1 minute. Thus, the user can click once to obtain a first result, and if he is willing to wait for a longer time with the aim of improving the results, he can click again. Furthermore, during the running of the reorganization algorithm, the user can still navigate in the cube. In this way, we hope that the reorganization will not distract (or delay) the user during his exploration process.

As previously mentioned, this interface can use a 3D stereoscopic screen. In this user study, we used the 3D hardware in all of the tasks. Our objective was not to discuss the interest of this immersive 3D setting (see [5] for more details).

---

1. The visual representation of two measures is performed with a double pyramidion: The height of the upper (respectively lower) pyramidion represents the first (respectively the second) measure. See [5] for more details.

---

Fig. 8. Visual comparison of the results on cube 4 for each of the tested methods. This cube has three dimensions (Store size, Time, and Product), two measures (Unit sales and Customer count), and 3,864 cells (see Table 2). Each method was allocated the same number of individual evaluations (6,000), except for the Level-BEA+GA hybrid approach (3,000). The best results are obtained by the GA, followed by the HC (which does not finely tune the obtained solution). The methods that are based on the BEA have a tendency to agglomerate the cells with large values. The methods that are based on the level by level approach suffer from the choices that are imposed from one level to the other. These methods do not get close enough to an optimum.
a complete discussion); our objective was instead to concentrate on the reorganization.

6.2 Overview of the User Study
The aim of this user study was to evaluate the integration of the reorganization methods in an OLAP user interface. Thus, we wanted to evaluate, from a user’s point of view, the studied framework with all of its elements (fitness function, genetic optimization, and integration in an OLAP user interface). An optimization may have good performances but it does not necessarily imply that the user performances will also be good because many factors can influence them.

To achieve this goal, we used the VR4OLAP interface, which was integrated with our GA, and we defined two tasks to be solved. These tasks were related to similarity detection (see the next section). For each task, we tested two methods, i.e., VR4OLAP with and without the GA-based reorganization, which will be subsequently denoted by VR4OLAP-R and VR4OLAP-no-R. For this last method, we ordered the members of a given level according to their alphabetical order (and with their hierarchical constraints).

For this study, 23 users were recruited (14 men and 9 women, aged from 20 to 35 years old, with an average of 23). These users were students at college level who attended courses on OLAP and business intelligence. They did not know the experimenters. Each user performed four trials (2 tasks × 2 methods). For each trial, the responses of the users were evaluated using the time to answer and the quality of the answer. Statistical tests were performed with a mixed effect model to possibly accept or reject the hypotheses (see the next section).

6.3 Tasks and Hypotheses
Two tasks were defined:

- T1: this task comprises finding, in a cube’s dimension, two members that have the same behavior with respect to the measure (i.e., two “slices” of the cube that are similar).
- T2: this task comprises finding, in a cube’s dimension, a member that has a different behavior than the others, i.e., an outlier.

For these two tasks, the user can use all of the possible interactions that are provided by our interface. The most interesting interactions for these tasks are the navigation around the cube (to obtain several possible viewpoints) and the dynamic adjustment of a scale factor that determines the spacing that exists between the displayed cells.

For each task and for each method, the answer given by the user was recorded as well as the time that was needed to answer. The correctness of an answer was evaluated using a quality measure that was defined for each task (see the next section). The effect of the selected method on the response time or quality was analyzed using a mixed effect model [36]. The hypotheses that we wanted to test had the following general expression:

\[ H_0: Mean_{VR4OLAP-R} = Mean_{VR4OLAP-no-R}, \]
\[ H_1: Mean_{VR4OLAP-R} \neq Mean_{VR4OLAP-no-R}, \]

which can be specialized for each task and each performance measure (time or quality), and will be further denoted by \(H_0(\text{time or quality, task})\) and \(H_1(\text{time or quality, task})\).

### Table 4

<table>
<thead>
<tr>
<th>Tasks</th>
<th>VR4OLAP-no-R</th>
<th>VR4OLAP-R</th>
<th>Prob.</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>0.66 (0.06)</td>
<td>0.80 (0.06)</td>
<td>0.1055</td>
</tr>
<tr>
<td>T2</td>
<td>0.56 (0.06)</td>
<td>0.78 (0.06)</td>
<td>0.0346*</td>
</tr>
</tbody>
</table>

The last column presents the probability of the F-test (mixed effect model), and a “*” indicates the statistical significance (Prob. < 0.05).

6.4 Design of the User Study
For T1, we created specific databases for which, in a given dimension, two members have similar measure values for the two other dimensions. For T2, we created databases in the following way: for a given dimension, several clusters of members with similar behavior were created. An “outlier” member, which did not belong to any of the clusters, was added to these members. These data cubes had similar characteristics (from 3 to 15 members per dimension). The users were not allowed to use the Roll-Up or the Drill-Down operators because developing/shrinking hierarchies would have changed the task to be solved from one experiment to the other (displayed data would have been different). For each task, we defined several databases that had a similar level of difficulty, to avoid any learning effect. The orders of the tasks, methods, and databases were also randomized.

The quality of an answer can be evaluated with a scalar value: when the user does not answer, the quality equals 0. When he gives the correct answer, the quality equals 1. Then, various degrees of correctness can be computed between 0 and 1. For T1, we use the similarity between the two members given by the user. For T2, we consider the cluster to which the answer belongs. We use the inverse of the cluster size to evaluate the quality of the answer. If the size is 1, then the correct answer was found because the outlier is a singleton. If the size is 5, for example, then the quality equals \(\frac{1}{5}\) (i.e., the larger the cluster, the worst the quality of the answer).

A preliminary questionnaire was filled out by the users to check basic inclusion criteria (their knowledge level of OLAP). Each user was trained on the user interface for 10 minutes.

The apparatus that was used in this study is the same hardware as mentioned previously, in Section 5.

6.5 Results and Discussion
The results are presented in Table 4. With regard to the response times, we observe a significant difference between the two tested methods: the reorganization helps users to answer the two questions faster \(H_1(\text{time, } T_1)\) and \(H_1(\text{time, } T_2)\) accepted. The results on the quality are significant for T2, where the reorganization leads to better results \(H_1(\text{quality, } T_2)\) accepted. For T1, the quality measures suggest a trend in favor of VR4OLAP-R, but more trials would be necessary to obtain significance.
These results can be interpreted as follows: The use of the reorganization eases the exploration of the data cube. Such an improvement is significant for the response times, which are almost halved when using the reorganization. Additionally, the reorganization improves the mean quality of the answer. Thus, we argue that the improvement that was measured in Section 5 using the homogeneity function corresponds also to an improvement in terms of visualization and users performances (at least for the tested tasks). A factor that might have contributed to these positive results is the use of a 3D representation. Such a representation can create occlusions, and the reorganization certainly helps to remove them. With fewer occlusions, the visualization becomes clearer and decisions are easier to make. We think that this influence exists but is limited in our experiment, for two reasons. First, the user can reduce the occlusions by dynamically modifying the spacing between the cubes that represent the values of the measure. Additionally, the databases were such that, even with occlusions, the visible part of the visualizations were sufficient to answer to the questions. In our future work on reorganization, we would like to know whether the improvement of performances in 2D would be as much important as the improvement observed in 3D.

Our study also confirms that the automatic reorganization of a visualization is appreciated by the users because the informal feedback from users was positive. Of course, the time needed for the reorganization was not included in the user response time, but no users have complained about the additional time that was needed to reorganize the cube. It appears that waiting for 1 minute was acceptable, considering the gain in user performance. Nevertheless, we believe that shortening further the running time is an important perspective.

7 CONCLUSIONS AND PERSPECTIVES

In this paper, the hierarchical reorganization of OLAP data cubes was studied to improve the visual representation of such cubes and to ease the knowledge discovery process. A new GA was proposed for the reorganization of k-ary trees, in which each tree corresponds to a hierarchical dimension in an OLAP data cube. This GA optimizes a visual homogeneity fitness function using standard genetic operators such as those defined for the traveling salesmen problem. We studied the parameters of this algorithm to obtain a performance/time ratio that is acceptable for an ONLINE analysis. A heuristic algorithm, which is inspired by the BEA, was studied to perform a level by level classification. The results have suggested that a hybridization of the GA approach. This goal can be accomplished in several ways. The results have suggested that a hybridization of the GA could be interesting. More precisely, we would like to study how to integrate BEA (or perhaps another method) as a heuristic-based mutation operator in the GA. In addition, an interesting property of GAs is their ability to be parallelized. Therefore, we would like to study how the fitness function, which has the highest computational cost, could be implemented on a parallel architecture, possibly with GPUs because the definition of the homogeneity function comprises fine grain level computations. Additionally, we are interested in using this hierarchical reorganization method in another domain. We would like to use our GA in the context of microarray data visualization and to compare it with existing approaches, including optimal approaches such as in [17].

ACKNOWLEDGMENTS

The authors would like to thank Julie Latreille from the CER.I.E.S. for her help in the design and performance of the statistical analyses, and the anonymous reviewers for their useful comments on this work.

REFERENCES


Sebastien Lafon received the master's degree in computer science from the University François-Rabelais of Tours, France.

Fatma Bouali received the PhD degree from the University of Paris-Sud. She is currently an associate professor of computer science at the University of Lille2 in France. Her research interests include visualizations for business intelligence, interactive exploration of databases, and visual data mining.

Christiane Guinot received the doctorate in biomathematics science (PhD degree) at the University of Paris in 1982, and the habilitation degree in computer science (DSc degree) at the University François-Rabelais of Tours in 2003. From 1982 to 1986, she completed two post-doctoral positions in medical statistics and epidemiology at the Gustave-Roussy Institute in France and at the Showa University School of Medicine in Japan. She was elected as a member of the International Statistical Institute in 2002. She was elected president of the French Statistical Association in 2005.

Gilles Venturini is a professor of computer science at the University François-Rabelais of Tours in France. His main research interests include visual data mining, virtual reality, 3D data acquisition and biomimetic algorithms (genetic algorithms, artificial ants). He is the coeditor in chief of the French New IT Journal (Revue des Nouvelles Technologies de l’Information) and was recently elected as a president of the French Data Mining Society (Extraction et Gestion des Connaissances).

For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/publications/dlib.