# A Novel Method for Selecting and Materializing Views based on OLAP Signatures and GRASP

Andresson da Silva Firmino[1], Rodrigo Costa Mateus[1], Valéria Cesário Times[1], Lucidio Formiga Cabral[2], Thiago Luís Lopes Siqueira[3], Ricardo Rodrigues Ciferri[4], Cristina Dutra de Aguiar Ciferri[5]

[1] Informatics Center, Federal University of Pernambuco, 50733-970, Recife, PE, Brazil
{asf2,rcm3,vct}@cin.ufpe.br
[2] Informatics Department, Federal University of Paraíba, 58035-000 , João Pessoa, PB, Brazil
lucidio@di.ufpb.br
[3] São Paulo Federal Institute of Education, Science and Technology, 13565-905, São Carlos, SP, Brazil
prof.thiago@ifsp.edu.br
[4] Department of Computer Science, Federal University of São Carlos, 13565-905, São Carlos, SP, Brazil
ricardo@dc.ufscar.br
[5] Department of Computer Science, University of São Paulo, 13560-970, São Carlos, SP, Brazil
cdac@icmc.usp.br

**Abstract.** Although the materialization of views reduces the execution time of OLAP queries, the materialization of a large number of views may exceed computer storage thresholds. Thus, given a certain storage cost threshold, there is a need for selecting the best views to be materialized, i.e. views that fit the storage requirements and provide the lowest response time to process OLAP queries. Several solutions have been proposed in the literature to solve this problem. However, most studies have adopted strictly greedy or purely random approaches. Also, most of them do not encompass the entire cycle of execution of multidimensional analysis, or do not specify and implement the whole cycle of multidimensional query execution. In this article, we address these issues by proposing a novel method for selecting and materializing views based on OLAP signatures and GRASP (Greedy Randomized Adaptive Search). On the one hand, using OLAP signatures and their relationships with descriptions of the data cube, we are able to identify which views should be materialized for being more beneficial to the user query processing. On the other hand, using GRASP allows us to define a hybrid method, which traverses the solution space in a comprehensive manner as performed in purely random approaches, while examines only the regions of the search space with a great concentration of good solutions generated by a greedy approach. GRASP was compared to other VSP algorithms, namely Pick by Size (PBS) and Ant Colony Optimization (ACO), and performance tests indicated that compared to PBS, the proposed method obtained a time reduction of about 20.4% in query processing. In addition, GRASP was more scalable than PBS, since it selected and materialized a smaller set of views, even when there was a wide range of possible views to be chosen. Also, GRASP obtained nearly the same query runtime of ACO (i.e. a small performance loss of about 2.84% was obtained by GRASP), but a shorter time for the selection of views than the ACO algorithm (i.e. a gain in processing time of about 77% was produced by GRASP).

Categories and Subject Descriptors: H. Information Systems [**H.m. Miscellaneous**]: Database Administration

Keywords: Data Warehouse, Materialized Views, Performance Evaluation

## 1. INTRODUCTION

Data Warehouse (DW) and On-Line Analytical Processing (OLAP) are widely used to aid the understanding of corporation data, aiming at increasing the productivity in decision-making processes that are supported by business applications. A DW is a multidimensional database that stores subject-oriented, integrated, time-variant and non-volatile data, and is often modeled through a star schema composed of fact and dimension tables. While fact tables store numeric measures of interest, dimension tables contain attributes that contextualize these measures. Also, attributes of a dimension may have a relationship with other attributes of the dimension through hierarchies. Multidimensional data of a DW are organized in different levels of aggregation. A lower level has detailed data, a higher level has very concise data and several intermediate levels represent increasing degrees of aggregation. All levels together compose a data cube that allows the analysis of numerical measures from different perspectives. For example, in a retail application, we can examine: (i) revenue per product, per customer city, per supplier (lower level), (ii) revenue by customer country and product category, revenue per year (intermediate levels), and (iii) total revenue (higher level). OLAP software enable the analytical processing of DW's multidimensional data according to different levels of aggregation.

In DWs, materialized views store precomputed aggregated data to eliminate overhead associated to expensive joins or aggregations required by analytical queries. Thus, given a certain storage cost threshold, there is a need for selecting the best views to be materialized, i.e. views that fit the storage requirements and provide the lowest response time to process OLAP queries. Several solutions have been proposed in the literature to solve this problem, that has been referred to as views selection problem (VSP), such as algorithms for views selection and the specification of costs and of selection criteria. Most studies do not encompass the entire cycle of execution of multidimensional analysis, or do not specify and implement the whole cycle of multidimensional query execution. Also, most of them have adopted approaches that are strictly greedy (e.g. [Chan et al. 2001; Nadeau and Teorey 2002; Aouiche et al. 2006; Vijay Kumar et al. 2011]) or purely random [Kalnis et al. 2002; Phuboon-ob and Auepanwiriyakul 2007; Derakhshan et al. 2008].

In this article, we define OLAP signatures as a dataset that is extracted from the historical records of OLAP queries previously issued by users. They characterize, throughout time, the behavior of users to analyze multidimensional data cubes. Using these signatures and their relationships with descriptions of the data cube, we propose a mechanism that is the main contribution of this article, and that encompass the entire cycle of multidimensional analysis, ranging from query execution to view materialization. We use these signatures for identifying characteristics of the views that will be analyzed by our proposed VSP algorithm.

Another contribution of this article is the specification of a novel VSP algorithm based on the metaheuristic GRASP (Greedy Randomized Adaptive Search) [Feo and Resende 1995]. Our hybrid approach traverses the solution space in a comprehensive manner as purely random approaches do, however it examines only regions of the search space that contain a great concentration of good solutions that were previously generated by a greedy approach. Semi-greedy solutions are generally more advantageous than a strictly greedy solution because they are expected to avoid local minimum values. Also, although GRASP has been successfully applied to several case studies [Festa and Resende 2009], it has not been used to solve VSP so far.

We validated our VPS algorithm through performance tests that investigated the benefits of using OLAP signatures together with the GRASP metaheuristic for selecting and materializing views. We compared GRASP to other VSP algorithms, namely Pick by Size (PBS) [Shukla et al. 1998] and Ant Colony Optimization (ACO) [Song and Gao 2010], and concluded that processing multidimensional views selected by GRASP provided a shorter query response time than PBS and nearly the same query runtime of ACO, but a shorter time for the selection of views than the ACO algorithm. In addition, GRASP was more scalable than PBS, since it selected and materialized a smaller set of views, even when there was a wide range of possible views to be chosen.

The remainder of this article is organized as follows. Section 2 describes the basic concepts used throughout the article. The contributions of the article are described in Sections 3 to 5, as follows. Section 3 lists the main reasons for having used the concept of OLAP signatures in this work and presents a conceptual model schema to represent information concerning these signatures, Section 4 details the novel method used for selecting and materializing views that is based on OLAP signatures and the GRASP metaheuristic, and Section 5 presents and discusses performance results. Section 6 surveys related work, while Section 7 concludes the article and highlights future work.

## 2. THEORETICAL FOUNDATION

In Section 2.1, we discuss the importance of using signatures in OLAP environments that motivated us to design a conceptual model schema for storing OLAP signatures. In Section 2.2, we describe the VSP that is tackled by our novel VSP algorithm. In Section 2.3, we outline the metaheuristic GRASP and the Ant Colony based optimization approach, which refer to the VSP problem and were used in the experimental work reported in Section 5.

### 2.1 OLAP Signatures

The user profile is a collection of data that describes the characteristics of a given user and corresponds to the signature of an individual, as the profile is the explicit digital representation of the user identity [Alfonseca and Rodríguez 2003]. Signatures represent or identify individuals. The generation of user profiles includes defining and maintaining signatures to determine, throughout time, the behavior of a given individual in a given domain or activity of interest [Hildebrandt et al. 2008]. In OLAP systems, user profiles are built from the analysis of multidimensional queries that were previously executed. Thus, we define OLAP signatures as a set of information extracted from the historical records of OLAP queries issued by users for specific purposes.

### 2.2 The Views Selection Problem

Recent studies show the importance of partial aggregation of multidimensional data to speed up the execution of OLAP queries [Khan and Aziz 2010]. However, the materialization of all possible views of a data cube is not feasible for the following reasons. Firstly, the materialization of each view has a cost that is affected by several issues, such as storage space and processing time for materializing views. Secondly, the number of possible views is usually very large and prohibitive. It is given by the combination $\prod^{d}{}_{i=1} n_i$, where $n_i$ is the number of levels per hierarchy of the dimension $i$, and $d$ is the number of dimensions of the data cube.

Due to the impossibility of having a complete data materialization, it is crucial to define which views should be materialized and how to select them. This definition has been refered to as views selection problem (VSP) [Khan and Aziz 2010], which is an optimization problem with a set of possible views, namely $V$, of a data cube $C$. It is essential to find the subset $M \in V$ that maximizes the benefit (or minimize the cost) of queries executed on $C$. In Section 6, we survey solutions proposed in the literature for solving this problem, and compare our proposed method with these related work.

### 2.3 The GRASP and Ant Colony Metaheuristics

There has been a growing interest in problem solving systems based on Greedy Randomized Adaptive Search Procedure (GRASP)[Feo and Resende 1995; Festa and Resende 2009]. Such systems are based on a metaheuristic for combinatorial optimization that consists of an iterative process with two phases: (i) a construction phase, in which a feasible solution is produced; and a (ii) local search phase, in which a local optimum in the neighborhood of the previously constructed solution is obtained. This process implies in the execution of two procedures until a certain stop criterion is reached. Firstly, the construction phase produces a feasible solution by creating a list with the best candidates using a greedy algorithm, such that one of its elements is randomly chosen and added to the current partial solution. As a result, this solution is a semi-greedy solution, i.e. a randomized greedy solution. Secondly, the local search phase obtains a local optimum in the neighborhood of the constructed solution, then updating the best solution found so far. For GRASP stop criterion, it is possible to use a certain number of executions or a particular limit of execution time. Our main motivation for applying the GRASP metaheuristic to the VSP problem involves the use of a probabilistic solution construction phase to improve the quality of the GRASP initial solutions. Then, for a given searching space, the GRASP local search phase allows the analysis of a good previous solution, by searching its neighborhood for another solution that is equivalent or better than the earlier.

The Ant Colony Optimization (ACO) is a metaheuristic that has been applied to optimization problems [Dorigo and Blum 2005] and to the VSP [Song and Gao 2010]. The ACO algorithm receives as input parameters: a list of views, the space available for materialization, the pheromone evaporation rate, the number of ants per vertex, the maximum number of attempts without improvements and the amount of pheromone deposited by an ant along a given path. This last parameter denotes the total size of all views given as input to the ACO algorithm. To solve VSP using ACO, ants build routes by determining if a given view should be added to the solution or not. For each path built, a quantity of pheromone is mapped to it. This pheromone information is used to guide the paths construction tasks of other ants. In addition, the pheromone evaporation may occur on the paths to avoid a premature convergence of the algorithm in a suboptimal region.

## 3. A CONCEPTUAL MODEL SCHEMA FOR REPRESENTING OLAP SIGNATURES

In this section, we define how characteristics of OLAP queries can be represented by OLAP signatures. The information concerning OLAP signatures proposed here characterizes, throughout time and for a given user, the kind of queries submitted by the user to a particular data cube. As a result, the most significant elements of the data cube are identified. Acquiring such knowledge aids the process of view selection, allowing us to choose the most beneficial views associated to the scope of previous queries submitted by the user. Another goal is to timely provide necessary information for the materialization of the chosen views. For this, we propose a representation for OLAP signatures and a conceptual model schema to associate OLAP signatures with elements of the data cube.

We propose that OLAP signatures be represented as shown in Figure 1 and described as follows. Each user query is related to a single OLAP signature. Also, each signature contains the following information: (i) a data cube, characterized by an identifier, a description and the name of the fact table whose data were used for generating the OLAP cube; and (ii) a set of combinations of levels among different dimensions, such that each existing combination represents a view (i.e. a vertex) that is described by an identifier, estimated number of tuples of the view, and size in bytes of each tuple of the view. Each vertex is a combination of levels of distinct dimensions.

Figure 1 shows the proposed conceptual model schema, in which each vertex represents information of a given data cube and is composed of several levels of aggregation. These levels have a precedence relationship with other levels to model the data cube hierarchies, and each of them has the following properties: an identifier, description, column name of the dimension table that is associated with this level, column data type (e.g. integer or double), column size in bytes, name of the dimension table associated with this level, a set of names of dimension tables used in the join operations and a set of join conditions used for accessing the level. Also, each level is related to a given data cube, which can have several measures. Each measure is described by an identifier, column name of its fact table, column data type, column size in bytes, and the numerical expression used for computing it. Therefore, information on measures, cube, levels and vertices are loaded by multidimensional data extractor algorithms discussed in Section 4.1, while query and signature are used to record historical information on the profiles of OLAP users as outlined in Section 4.2.

According to the conceptual schema of Figure 1, an user query is described by a MDX (MultiDimensional eXpressions) expression [Whitehorn et al. 2005] and the date in which the user requested the query execution. Our work is based on MDX because this multidimensional query language is a well-known standard for processing OLAP queries. Data related to the submitted user query are

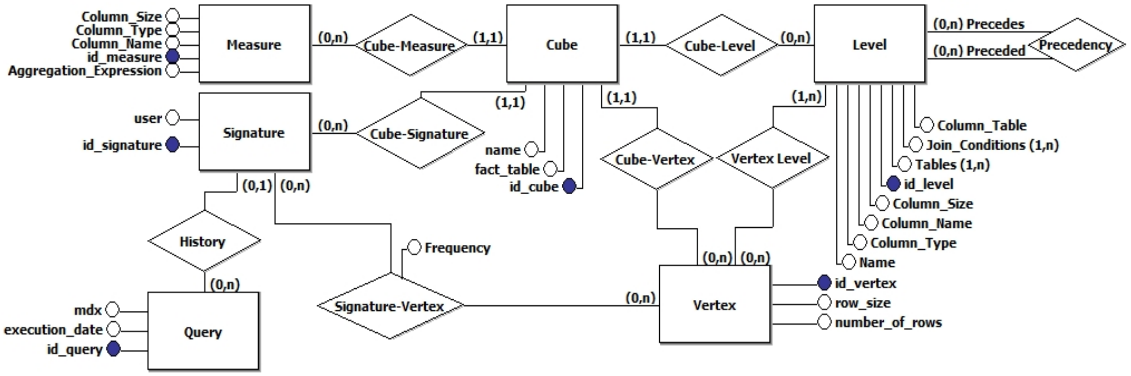


Fig. 1. A conceptual model schema for OLAP Signatures.

also recorded, including user query expression, query execution date and the relationship between the query signature and the view with the minimum cost (e.g. view with the smallest size) that may be used to answer the user query. In our conceptual model schema, we store information about sizes in bytes, since storage costs are given by multiplying the number of lines by the size of these lines. This computation indicates the size of a view and is one of the aspects used to select views to be materialized according to the storage space available. The frequency information of views is also used to select views to be materialized as further explained in Section 4.2 and is incremented when an instance of the relationship between the query signature and the view with the smallest size is identified.

## 4. A METHOD FOR SELECTING AND MATERIALIZING VIEWS

In this section, we introduce the proposed method for selecting and materializing views, which is composed of four phases: (i) extract data and metadada from a DW, from its corresponding star schema, and from a data cube schema, and then, compute all possible vertices for this cube (Section 4.1); (ii) record information about the user OLAP queries submitted so far and build the corresponding OLAP signatures (Section 4.2); (iii) select the more beneficial vertices (i.e. views) for the execution of user OLAP queries (Section 4.3); and (iv) materialize the chosen vertices (Section 4.4).

### 4.1 Extracting Multidimensional Data

This phase aims to extract information from a mapping schema between a DW and an OLAP cube and to create and store the entities *Cube*, *Measure*, *Level* and *Vertex* shown in the conceptual metamodel of Figure 1. This is performed by our multidimensional data extractor algorithm (Algorithm 1), which works as follows. It receives as input a mapping *schema* from which the conceptual metamodel's entities of Figure 1 are created, instantiated and persisted. In line 1, the procedure *ReadCubes* is called to obtain a set of instances of the entity *Cube* from *schema*, and to store them in the variable $setOfCubes$. Next, in line 3, each instance $a\_cube$ of $setOfCubes$ is stored. In line 4, a set of instances of the entity *Measure*, which is defined in *schema* for $a\_cube$, is read and written in the variable $setOfMeasures$, and in line 5, the instances of $setOfMeasures$ and their corresponding relationships with $a\_cube$ are saved. In line 6, it is obtained the set of dimensions defined for $a\_cube$ from *schema*, which is stored in the variable $setOfDimensions$. In line 7, the accumulator $totalSetOfLevels$, which aims to store all sets of levels of each dimension, is initialized. Then, in line 9, for each dimension $a\_dim$ of $setOfDimensions$, it is obtained a set of hierachies denoted by $setOfHierarchies$ from *schema*. Next, in line 10, the accumulator $setOfLevelsByDim$ is created to store the set of levels of $a\_dim$, and is initialized to an empty set. For each hierarchy (indicated by $a\_hierarchy$ in line 11) of $setOfHierarchies$, a set of instances of the entity *Level* are read from *schema* and stored

```
Algorithm 1: ExtractMultidimensionalData (schema)
Input: a mapping schema between a DW and an OLAP cube
Output: Void
1   setOfCubes ← ReadCubes(schema);
2   FOR (a_cube: setOfCubes) DO
3       StoreCube(a_cube);
4       setOfMeasures ← ReadMeasures(a_cube,schema);
5       StoreMeasures(setOfMeasures, a_cube);
6       setOfDimensions ← ReadDimensions(a_cube,schema);
7       totalSetOfLevels ← ∅;
8       FOR (a_dim : setOfDimensions) DO
9           setOfHierarchies ← ReadHierarchies(a_dim,schema);
10          setOfLevelsByDim ← ∅;
11          FOR (a_hierarchy : setOfHierarchies) DO
12              setOfLevels ← ReadLevels(hierarchy, schema);
13              FOR (a_level : setOfLevels) DO
14                  precedents_levels ← GetPrecedents(a_level, a_hierarchy);
15                  StoreLevel(a_cube, a_level, precedents_levels);
16              setOfLevelsByDim ← setOfLevelsByDim ∪ setOfLevels;
17          totalSetOfLevels ← totalSetOfLevels ∪ setOfLevelsByDim;
18      setOfVertexes ← GetVertexes(a_cube, totalSetOfLevels);;
19      StoreVertexes(setOfVertexes, a_cube);
```

```
Algorithm 2: GetVertexes (a_cube, totalSetOfLevels)
Input: an OLAP cube and the set of all levels of all dimensions
Output: set of vertexes successfully created
1   setOfVertexes ← ∅;
2   levelsOfCP ← CartesianProduct(totalSetOfLevels);
3   FOR (setOfLevels: levelsOfCP) DO
4       row_size ← 0;
5       number_Of_rows ← 1;
6       FOR (a_level: setOfLevels) DO
7           number_Of_rows ← number_Of_rows * GetNumberOfRows(a_level);
8           row_size ← row_size + GetRowSize(a_level);
9           row_size ← row_size + GetPrecedentsSize(a_level);
10      row_number_estimated ← GetEstimation(number_Of_rows);
11      row_size ← row_size + GetMeasuresSize(a_cube);
12      a_vertex ← NewVertex(row_size, row_number_estimated, setOfLevels);
13      setOfVertexes ← setOfVertexes ∪ a_vertex;
14  RETURN setOfVertexes;
```

in the variable *setOfLevels* (line 12). For each instance *a_level* of *setOfLevels*, their respective precedent levels in the hierarchy are obtained in line 14. Then, each level (*a_level*) and their respective relationships with *a_cube* are recorded (line 15). In line 16, the set of levels of each hierarchy *a_hierarchy* is added to *setOfLevelsByDim*, while in line 17, the set of levels of each dimension (i.e. *setOfLevelsByDim* of each *a_dim*) is added to *totalSetOfLevels*. Once stored the entities *Cube*, *Measure* and *Level*, and having the whole set of levels of all dimensions (denoted by *totalSetOfLevels*), the algorithm calls the procedure *GetVertexes* (Algorithm 2) to generate the set of instances of the entity *Vertex* that represent all possible views for a given cube (i.e. *a_cube*), and assigns this set to *setOfVertexes* (line 18). Finally, in line 19, all instances of *setOfVertexes*, their respective relationships with *a_cube*, and all levels that compose each view are stored.

The *GetVertexes* algorithm (Algorithm 2) works as follows. It takes as input an instance *a_cube* and all instances of the entity *Level* (indicated by *totalSetOfLevels*). In line 1, *setOfVertexes*, which represents a set of instances of the entity *Vertex*, is initialized to an empty set. It aims to store the vertices that will be generated and returned by the algorithm. In line 2, a procedure is called to calculate the cartesian product among all sets of *totalSetOfLevels*. From this computation, a collection of sets of instances of the entity *Level* is returned. To generate an instance of the entity *Vertex*, the following information is needed: the line size (i.e. attribute *row_ size* of the entity *Vertex*), the estimated number of rows (i.e. attribute *row_ number_ estimated*) and the set of levels that compose the vertex (i.e. $setOfLevels$). The line size is given by the sum of the attribute *column_ size* of a given $a\_level$, which denotes the memory size in bytes of $a\_level$, with the sum of *column_ size* of each level that preceeds $a\_level$, and plus the sum of each *column_ size* of each measure of $a\_cube$ (lines 7 to 9). In line 10, the *row_ number_ estimated* is obtained through the computation of an estimated number that is based on the number of tuples of the fact table that is related to a given cube $a\_cube$ and on the number of tuples resulting from the multiplication between the number of rows (i.e. number of members) of each level of $setOfLevels$. As each instance $a\_vertex$ is created, it is added to $setOfVertexes$ (line 13). Finally, in line 14, the set of instances of *Vertex* is returned.

### 4.2 Recording User OLAP Queries and Building OLAP Signatures

This phases aims to record information about the user OLAP queries submitted and build the corresponding OLAP signatures. It includes two subphases. The first one consists in the specification of the association between the user and a data cube, which results in the signature computation and in the determination of the relationship between the signature and the OLAP cube. The second sub-phase consists in building the history of user OLAP queries and in tracking the levels involved in the queries. Therefore, it identifies the vertices that when materialized will be used to answer these queries. To achieve this, for each query submitted, the user responsible for its submission is identified and the data cube over which the query is executed is determined as well. Using the pair $(user, cube)$, the signature, to which the query belongs, is identified. Thus, the OLAP query is stored, and its relationship with the OLAP signature is identified.

Many vertices (i.e. views) can be used to solve a given user query. Once these vertices are known, we must identify the vertex of smaller size, i.e. the vertex of minimum cost to answer the query in question. Then, we register the occurrence of the relationship between the query signature and the vertex of minimum cost. Finally, if the occurrence of the relationship has already been registered, we only increase the frequency of the attribute that denotes this relationship.

### 4.3 Selecting Views

The objective of this phase is to select the vertices considered as more beneficial for processing the OLAP queries, based on the data associated with the OLAP signatures generated in the previous phases. The selection criteria are described in Section 4.3.1 and the optimization algorithm adopted for selecting the views is explained in Section 4.3.2.

4.3.1 *The Selection Criteria Used.* Data stored during the first and second phases provide information on the vertices of a cube associated with a given OLAP signature. This information is read and handled as a set of quadruple *(id_ vertex,size,benefit,density)*, and is given as input for the selection process. The first element of this quadruple is the identifier of a vertex, which is used after the selection phase for identifying the vertices selected for materialization. The second element is the size of the vertex, which is obtained by multiplying the attributes *row_ size* and *number_ Of_ rows* of the entity *Vertex* of Figure 1. The third is the benefit given by the vertex in the case of being materialized to assist in the execution of queries related to a given signature. The calculation of the benefit of a vertex $v$ of a given signature $a$ is given by Equation 1, where *maximumFrequency(a)* denotes the highest frequency of all vertices of the signature *a*. The maximum frequency is used in this computation to normalize the benefit calculated. The fourth element of the quadruple is the density that is obtained by the ratio between the benefit and the size of the vertex. The density represents how valuable the materialization of the vertex is, because the greater the benefit and the smaller the size occupied by the vertex, the more valuable the vertex is to run a given query. Therefore, the density

```
Algorithm 3: GRASP ($\varphi$, $\delta$, $\gamma$, $\theta$, $\vartheta$, $\tau$)
Input: a list of vertices ($\varphi$), amount of space available for materialization ($\delta$), number of algorithm iterations
       used as a stop criterion ($\gamma$), greedy rate to define the size of the LRC ($\theta$), greedy rate to define
       the size of the LRC in local search ($\vartheta$), number of exchanges to be made in local search ($\tau$)
Output: best solution
1  bestSolution ← $\theta$;
2  k ← 1;
3  FOR (k: $\gamma$) DO
4        $\varphi'$ ← $\varphi$;
5        solution ← ConstructionPhase($\varphi'$, $\delta$, $\theta$);
6        solution ← ImprovementPhase($\varphi'$, $\delta$, $\theta$, $\tau$, solution);
7        UpdateSolution(solution, bestSolution);
8  RETURN bestSolution;
```

is used as the criterion for selecting the vertices by our optimization algorithm.

$$\textit{benefit(v,a)} = \textit{frequency(v,a)} \;/\; \textit{maximumFrequency(a)} \tag{1}$$

4.3.2 *The Optimization Algorithm Proposed.* The proposed optimization algorithm based on the GRASP meta-heuristic (Algorithm 3) works as follows. It receives as input a list of the quadruples described in Section 4.3.1 and sorted by density. Each quadruple of this list represents a vertex. It is noteworthy that the algorithm will process only the vertices whose benefit is greater than zero. In line 1, the best solution is initialized to an empty set of vertices, i.e. at the beginning of the algorithm, the benefit of the best solution is zero. In lines 2 to 7, the GRASP iteractions are performed. In line 3, the variable $\varphi$ is copied into $\varphi'$ at each beginning of iteration, so that the list of vertices given as input to the *ConstructionPhase* procedure of line 5 remains unchanged. In line 5, a partial solution is obtained through the construction phase and is stored in the variable *solution*. Then, in line 6, through the improvement phase, a new solution, which is equal to or better than the previous solution, is obtained and stored in the variable *solution*. In line 7, the *UpdateSolution* procedure is called to compare the current best solution (*bestSolution*) and the solution obtained in the previous line (*solution*). It returns in *bestSolution* the solution with the best benefit. The benefit of a solution is given by the sum of benefits of each vertex that composes the solution. Finally, at the end of the iterations, in line 8, the best solution found so far, which consits in the best set of vertices for helping in the execution of queries of a given signature, is returned.

In each iteration of the GRASP algorithm, the *ConstructionPhase* procedure is called. The *ConstructionPhase* algorithm (Algorithm 4) aims at generating a good initial solution. In line 1, *Solution* is initialized to an empty set of vertices. In lines 2 to 9, iterations are performed while the size of the solution is not greater than the space available for materialization. Each iteration begins with the definition of the restricted candidates list, i.e. RCL. This list contains the elements that were best evaluated by the greedy approach, i.e. the vertices with higher densities. In line 4, it is verified whether there are elements in RCL that can be added to the solution. If RCL is not empty, the *RandomSelection* procedure, which randomly selects a vertex from RCL, is called in line 5. This randomness is the differential of GRASP, since the choice of the vertices that comprise the solution is based on a semi-greedy approach. Thus, solutions of good quality are generated due to greedy property, and diversified due to the random feature. In lines 6 and 7, the selected vertex is removed from $\varphi'$ and added to the solution. In line 10, the solution built at the end of the iteractions is returned.

The *CreateRCL* procedure called by the GRASP construction phase algorithm (line 3 of Algorithm 4) first initializes the RCL list, which is returned by this procedure as an empty set of vertices. Then, the values of minimum and maximum density for the set of vertices given by the first input parameter ($\varphi'$) of this procedure are computed. Through the multiplication between the greedy degree (i.e. $\theta$) and the difference between maximum and minimum densities whose product is subtracted from the

```
Algorithm 4: ConstructionPhase ($\varphi'$, $\delta$, $\theta$)
Input: a list of vertices ($\varphi'$), amount of space available for materialization ($\delta$),
       greedy rate to define the size of the LRC ($\theta$)
Output: best solution
Solution ← $\emptyset$;
WHILE (SizeOf(Solution) < $\delta$) DO
    RCL ← CreateRCL($\varphi'$, $\delta$, $\theta$);
    IF (NotEmpty(RCL)) THEN
        $\upsilon$ ← RandomSelection(RCL);
        $\varphi'$ ← $\varphi'$ - $\upsilon$;
        Solution ← Solution ∪ $\upsilon$;
    ELSE
        GO TO 10;
RETURN Solution;
```

```
Algorithm 5: ImprovementPhase ($\varphi'$, $\delta$, $\vartheta$, $\tau$, solution)
Input: a list of vertices ($\varphi'$), amount of space available for materialization ($\delta$), greedy rate to define
       the size of the LRC in local search ($\vartheta$), number of exchanges to be made in local search ($\tau$),
       solution built in the construction phase (solution)
Output: a similar or improved solution
1   numberOfTrials ← $\tau$;
2   WHILE (numberOfTrials > 0 ) DO
3       OutRCL ← SearchRCL($\varphi'$, $\vartheta$, bestVertices);
4       InRCL ← SearchRCL(Solution, (1 - $\vartheta$), worstVertices);
5       IF (NotEmpty(OutRCL)) THEN
6           IF (IsOdd(numberOfTrials)) THEN
7               swapOneToTwo(OutRCL, InRCL, $\varphi'$, Solution, $\delta$);
8           ELSE
9               swapOneToOne(OutRCL, InRCL, $\varphi'$, Solution, $\delta$);
10      numberOfTrials ← numberOfTrials - 1;
11  RETURN Solution;
```

maximum density, a greedy threshold is calculed. This threshold defines the elements of $\varphi'$ that will compose RCL. In fact, each vertex of $\varphi'$ is added to RCL if its density is not greater than this threshold and if its size added to the size of solution is not greater than the space available for materialization.

The second phase of GRASP, the phase of improvement (or local search) aims at refining the solution obtained in the construction phase in order to find better solutions. Given a solution $s$, the search for better solutions is made by visiting neighboring solutions in order to find among the neighbors, a better solution than $s$. The searching techniques used are $swapOneToTwo$ and $swapOneToOne$. The first is the exchange of one element of a given set by two other elements of another set, while the second technique is the exchange of one element of a given set by another element of another set. The permutations are performed over the vertices of the solution built in the constructive phase (i.e. *solution*) and the vertices that were not chosen to compose the solution (i.e. the vertices of $\varphi'$). Using $\varphi'$ and *solution*, two RCLs, $OutRCL$ and $InRCL$, are built respectively. $OutRCL$ consists in vertices of $\varphi'$ with better density, while $InRCL$ is composed of the vertices with lower density contained in *solution*. The goal is to switch low density vertices of *solution* for high density vertices that are out of *solution*. These lists are built by the $SearchRCL$ procedure that works similarly to the procedure for creating the RCL in the GRASP construction phase (algorithm 4). The differences are that the space restriction is not considered and a new parameter is used to indicate if the best or worst vertices are returned. The best vertices are those whose density values are found between the maximum density and the greedy threshold, while the worst vertices are those whose densities belong to the following boundaries: the greedy threshold and the minimum density value.

The GRASP *ImprovementPhase* algorithm (Algorithm 5) works as follows. In line 1, the number of trials is initialized to $\tau$ that denotes the stop criterion of this algorithm. In lines 2 to 10, a loop is executed while the number of trials is greater than zero. In each iteration of this loop, the $OutRCL$ (i.e. a list composed of $\vartheta$ vertices with the highest density values) and $InRCL$ (i.e. a list of $\vartheta$ vertices with the lowest density values or a list of vertices that are not the best (1 - $\vartheta$)) lists are created. In line 5, it is checked if there are vertices to be exchanged. Then, for exchanging vertices, swapping procedures are called according to the parity of the number of trials. These procedures verify whether: (i) the size of the solution resulting from a possible exchange of vertices does not exceed the amount of space available for materialization; and (ii) the benefit of vertices added to the solution is greater than the benefit of vertices removed from this solution. At the end of each iteration, in line 10, the number of trials is decremented by one. Finally, in line 11, the best solution found is returned.

### 4.4 Materializing Views

The last phase is the materialization of vertices chosen in the previous phase. The materialization of a vertex consists in creating a data table and loading this table with appropriate aggregated data. To build the table, information about the format of its columns format are needed. The table columns are composed of cube's measures and levels that compose the vertex. Once identified the measures and levels, the attributes $Column\_Name$ and $Column\_Type$ of the entities $Level$ and $Measure$ are used for enabling the construction of the table creation script. Once created the tables, they are populated with data originated from aggregations on measures of fact tables. These aggregations result from the clustering of measures with the combination of levels of the vertex. To achieve this, a selection script is built by using the attributes *column_name* and *aggregation_expression* of the entity $Measure$, and the attributes *column_name*, *column_table*, *tables* and *join_conditions* of the entity $Level$.

## 5. EXPERIMENTAL EVALUATION

This section is organized as follows. Section 5.1 describes the experimental setup that was used to evaluate the GRASP-based view selection algorithm proposed in this article. This experimental setup aims at assessing the benefits of using a small number of materialized views to answer analytical queries by giving as input to the optimization algorithm only those views with the minimal cost (see Section 3). Experimental tests were carried to define the best values for the GRASP configuration parameters and results derived from these tests are given in Section 5.2. The performance evaluation results are discussed in Section 5.3.

### 5.1 Experimental Setup

Experiments were conducted on a computer with 3.33 GHZ Intel Core i7 processor, 16 GB of main memory, 7200 RPM SATA II 2 TB hard disk, Windows Server 2008 R2 Standard, Mondrian 3.2.0 and PostgreSQL 9.0. Algorithms of selection and materialization of views described in this article were implemented in Java programming language to integrate them with Mondrian. We have chosen the PostgreSQL to store the signatures. We also adapted the schema and the workload of the Star Schema Benchmark (SSB) [OŠNeil et al. 2009] to increase the number of dimensions and levels and to have a greater number of queries associated to a greater number of diversified views. These adaptations were required firstly to measure the impact of executing varied and recurrent queries and then verify how many times a given view can be used to answer queries, and secondly to investigate the algorithm scalability and check its effectiveness when there is a wide range of possible views to be chosen.

Regarding the OLAP cube schema used in the experiments, it is available at $http : //www.cin.ufpe.br/\tilde{~}asf2/esquema.pdf$. This multidimensional data schema is composed of 5 dimensions (i.e. *Part*, *Supplier*, *Customer*, *Date* and *CommitDate*), 35 levels and 2 measures (i.e. *Lo_Revenue* and *Lo_SupplyCost*). This schema was created using SSB's database with scale factor

1. Data generation produced 6 million tuples in the fact table, and the algorithms 1 and 2 listed in Section 4.1 have yielded 14,175 possible materialized views, whose estimated total size is 6,69 TB.

The workload was based on 1,280 query submissions randomly chosen from a set of 32 MDX distinct queries. These queries include Q2, Q3 and Q4 queries from the SSB, i.e., those queries that can be translated to MDX without changing their original selectivity. The queries were designed to access a large number of diverse views, in order to investigate the ability of VSP algorithms on operating adequately even when there are several and diverse views available. The workload has queries that vary the number of views accessed (i.e. cardinality) as well as vary the uniqueness of the combination of levels involved (i.e. composability). As the queries were consecutively submitted, the historical user profile was then built and enabled for assessing the performance of materialized views. We grouped the queries according to the cardinality and reported them at *http://www.cin.ufpe.br/~asf2/ConsultasMDX.pdf*.

### 5.2 Parameters Tuning and Test Scenarios

The view selection algorithm proposed in this article receives as input data: (i) a set of views, where each view is represented by a quadruple of type *(id_vertex,size,benefit,density)*, and (ii) a set of GRASP configuration parameters. It is important to note that changing the GRASP's configuration parameter values affects the quality of the solutions generated by GRASP for a given input set of views. Therefore, we investigated the quality of solutions computed, varying the values of these parameters. After completing these preliminary tests, the following values of GRASP configuration parameters were seen as the best ones: $\{\theta = 0{,}6;\ \vartheta = 0{,}05;\ \gamma = 25;\ \tau = 10\}$. Further details on these test results are documented in *http://www.cin.ufpe.br/~asf2/GraspConfiguration.pdf*.

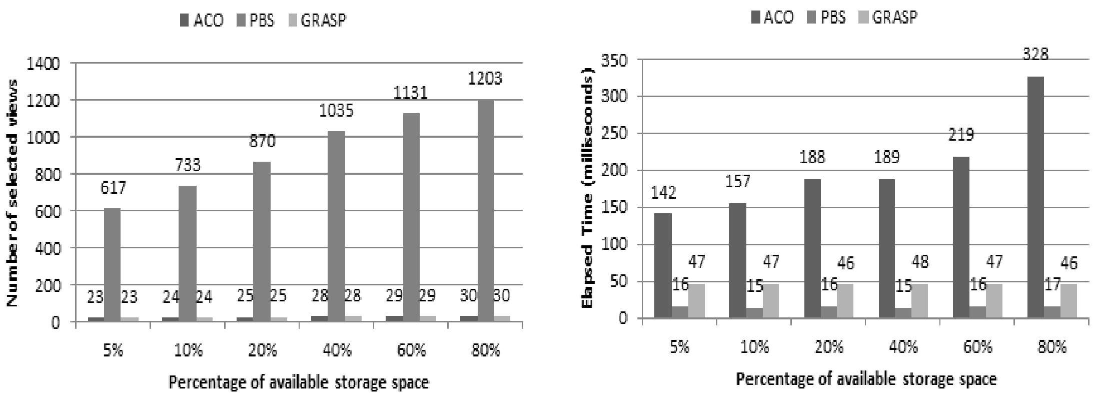
Similarly to the GRASP, for a input set of views, when changing the ACO's configuration parameter values, the quality of the solutions generated is affected. Therefore, we investigated the quality of the computed solutions by changing the values of these parameters and we noticed that the quality of the solution was not changed from the following values: {the pheromone evaporation rate $(\rho) = 0.93$; the number of ants per vertex$(\pi) = 1$; the maximum number of attempts without improvements$(\varepsilon) = 1$} and thus, these were seen as the best ACO's configuration values. Further information on these configuration tests are found in *http://www.cin.ufpe.br/~asf2/AntConfiguration.pdf*. It is important to note that no parameter settings for PBS was needed because it lacks input parameters.

A query profile defines a sequence of queries randomly chosen from a set of 32 distinct MDX query types. In the sequence, each MDX query type appears at least once, so that the total number of repetitions of each query type in the sequence is equal to the cardinality of the sequence. A query profile is the frequency distribution of MDX query types found in the sequence. For instance, in query profile 10_90, 10% of the MDX query types have 90% of the frequency, i.e. in a sequence composed of 100 queries and for 10 MDX query types, the frequency of one specific query type is equal to 90, while the total frequency of the other types of MDX queries is 10. Therefore, the purpose of using query profiles is to evaluate the response time in different scenarios of queries submissions, so that a set of queries may have frequencies of submission that are greater or less than another set of queries.

### 5.3 Performance Results

In performance tests, we materialized views using GRASP and two other VSP algorithms, called PBS (Pick by Size) [Shukla et al. 1998] and ACO (Ant Colony Optimization) [Song and Gao 2010], and compared the quality of generated solutions. We adopted as units of quality measurements, the response time of queries processed on materialized views, since they aim at reducing the execution time of analytical queries. Also, we show that the quality of views chosen by each algorithm is not related to the number of views selected nor to the algorithm's execution time for selecting views.

PBS is a fast algorithm, has broadly been used in comparative analysis, providing good results



(a) Number of available storage space

(b) Time spent to select views.

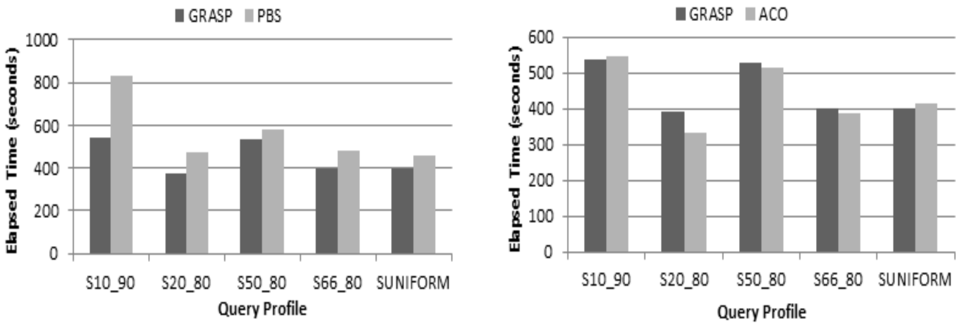
Fig. 2. Performance Evaluation between GRASP, PBS and ACO

[Kalnis et al. 2002; Lijuan et al. 2009; Khan and Aziz 2010] and has a high cardinality of selection because it uses a single criterion of selection that is the view size, while our GRASP proposal and ACO are more complex, and are expected to have a greater runtime of selection of views and a low cardinality of selection, once they adopt as selection criteria both the view size and the frequency of user queries, and they receive as input a reduced set of views to be selected. Note that the small number of selected views by GRASP and ACO did not reduce the storage costs of these views, once the space available for materialization was always the same for all three algorithms.

Experiments considered the following values of the space available for materialization (in GB): {0,25; 0,5; 1; 2; 3; 4} that represent the following percentages {5; 10; 20; 40; 60; 80}, respectively, on the total space of materialization of all views (i.e. 32 views occupying 5GB in total that were generated by algorithms 3, 4 and 5 of Section 4.3) given as input to GRASP and ACO as well. Figure 2(a) displays the number of views selected by the algorithms according to the percentages of space available for materialization. Clearly, GRASP and ACO selected much less views than PBS for the same storage space. Figure 2(b) shows the elapsed time spent by each algorithm to select these views.

Notably, PBS had the highest number of views selected with the 4 GB storage threshold. Therefore, we conducted another test using this threshold in order to compare the query processing using the views materialized by both the PBS and GRASP. We aimed at investigating if the selection of a large number of views would imply in better performance to process OLAP queries. All views selected by each algorithm were materialized and query profiles *10_90*, *20_80*, *50_80*, *66_80* and *UNIFORM* were used. The uniform profile means that all MDX query types have the same frequency. Also, GRASP was configured with the same values of parameters, and we used the same set of 32 distinct MDX queries that was used for generating historical queries that composed the OLAP signature. A sequence of 580 MDX queries was submitted, and we gathered the elapsed time to process all 580 queries according to each profile. For the GRASP, views were selected and materialized according to this signature, while for the PBS, views were selected according to their sizes. Each algorithm used its own set of selected and materialized views, and had the same execution sequences of queries.

Figure 3(a) shows the results, which indicated that executing queries using views created by GRASP is more efficient than executing queries using views selected by PBS, considering all query profiles. We also highlighted the time reduction provided by GRASP over PBS, which ranged from 8% to 35%. Considering the sum of the elapsed time to process the 2,900 queries (580 for each profile), there was also a 20% time reduction provided by GRASP over PBS. These results corroborated that materialized views selected by GRASP should be used to obtain a better query processing performance. Furthermore, they indicated that although we have used the storage threshold of 4 GB, so that the



(a) Runtime of Query Profiles (GRASP *vs.* PBS)

(b) Runtime of Query Profiles (GRASP *vs.* ACO)

Fig. 3. Runtime of Query Profiles

largest number of views is produced by PBS, PBS could not process OLAP queries faster than GRASP. Therefore, it means that the selection of a large number of views does not necessarily imply in a large time reduction to process OLAP queries.

To compare GRASP with a recent approach that also selects a small number of views, further experiments were conducted. These aimed at investigating whether GRASP would provide similar reduction in time for selecting views to be materialized and for processing queries over these views as well. Figure 2(a) shows that ACO and GRASP selected the same number of views. However, the elapsed time spent by GRASP to choose these views was shorter than the corresponding time of ACO, as shown in Figure 2(b). Also, these two search algorithms selected the same set of views, except for the case of 3GB storage threshold, in which the sets of views are different solely because of a single view. Thus, using the 3GB storage threshold, we compared the query processing using the views materialized by both the ACO and GRASP. Figure 3(b) shows the results, which indicated that executing queries using views created by ACO is a little more efficient than executing queries using views selected by GRASP, because considering the sum of the elapsed time to process the 2,900 queries (580 for each profile), there was a 2,84% time reduction provided by ACO over GRASP. Therefore, our tests have pointed out that ACO is slightly faster than GRASP in query execution, but implies in longer execution time for selecting views to be materialized than GRASP.

## 6. RELATED WORK

Several approaches have been proposed for solving the problem of VSP. The VSP algorithms can be classified into four main categories according to the applied optimization technique:

—Greedy Heuristics: they always select what seems to be the best choice at the moment, making a local optimum choice in the hope that this leads to the global optimum solution [Vijay Kumar et al. 2011; Chan et al. 2001; Nadeau and Teorey 2002; Shukla et al. 1998]. Their implementation are often easy and produce fast and efficient results. However, they have the disadvantage of generating a very limited number of solutions, i.e. for a given input dataset, the solution obtained is always the same. This occurs because greedy decisions, taken at the begining of the solution construction process, strongly constrain the possibilities of building the solution in later phases, and consequently, this impacts in the execution of restricted movements during the construction process, usually leading to sub-optimum solutions.

—Random Search Heuristics: they traverse the solution space in order to find good solutions by making pseudo-random movements [Derakhshan et al. 2008; Kalnis et al. 2002; Phuboon-ob and Auepanwiriyakul 2007]. Within this group, there is a subclass of bioinspired techniques that are based on the behavior of populations and aim to build an effective navigation approach for searching the solution space [Song and Gao 2010; de Albuquerque Loureiro 2006; Sun and Wang 2009; Li et al. 2010]. They avoid minimum and maximum local values and generate a large number of solutions. However, depending on the degree of randomness of the search movement, the resulting solutions may be widely dispersed and distant from the optimal value.

—Evolutionary Heuristics: they are based on concepts related to the biological evolution of individuals and their goal is to evolve previously identified solutions in order to find optimum (or sub-optimum) values [Lawrence 2006; Lee and Hammer 2001; Talebian and Kareem 2009; Yu et al. 2003; Zhang et al. 2001]. It has the advantage of providing much better solutions than other heuristics. However, some difficulties may be found for coding the problem and another limitation of these techniques is that they usually require a higher computational cost than the other heuristics.

—Hybrid Heuristics: they use a combination of the aforementioned techniques. The goal is to explore good properties of different techniques in order to solve problems more efficiently [Lin and Kuo 2004; Wang and Zhang 2005; Yuhang et al. 2010; Zhang et al. 2009].

—Model-driven Heuristics: are mostly based on mathematical models specifically designed for the problem of VSP [Dobbie and Ling 2000; Gou et al. 2006; Smith et al. 2004; Lin et al. 2007; Serna-Encinas and Hoyo-Montano 2007; Lijuan et al. 2009]. In some approaches, the acquisition of the optimal solution is guaranteed, but for this, the number of views should be small due to the solution building feasibility, making it impractical in real cases, where the number of views is very high [Yang et al. 1997; Theodoratos and Bouzeghoub 2000; Chirkova et al. 2006].

For the sake of simplicity, the VSP configuration used in this work is static (i.e. views are selected statically, materialized and maintained until changes are needed). In addition, the static approach seemed to be more appropriate in scenarios where one knows in advance the scope of OLAP queries, and one of our goals was to build the profile of these queries (i.e. OLAP signatures). Regarding the use of workload queries, our approach also differs from other related studies because: (i) Firstly, besides keeping information about OLAP queries that were previously executed, we also stored information about the data scheme used to process these queries; and (ii) Secondly, using a data model, we defined how the history of the OLAP queries is stored and subsequently, used to select and materialize views. To select a view to be materialized, we needed to estimate the number of lines of the view being analysed. We used the tuples estimation given in [Shukla et al. 1998], whose formula is Est $(v) = ne - ne * (1 - 1/ne)^{nf}$, where $nf$ is the number of tuples of the fact table and $n$ is the maximum number of rows of the view $v$. The value of $n$ was obtained by multiplying the number of distinct members that compose each level of the view $v$. Another diferential of our approach is that most of previous work have a coupling between the chosen data structure and the algorithm's logic of selection, while our approach is modularized, separating data structure issues from the adopted VSP logic.

## 7. CONCLUSION AND FUTURE WORK

This article proposed an algorithm for tackling the views selection problem (VSP) based on the GRASP metaheuristic. GRASP is a hybrid heuristic that combines the features of both greedy and random search heuristics and has been recognised in the literature as one of the most competitive metaheuristic approaches in terms of the quality of the generated solutions. In fact, the impact of using GRASP in the VSP has so far not been studied. In this article, we used as selection criteria both the space requirements for the views and the frequency of execution of OLAP queries over these views. In addition, we defined a conceptual model scheme that stores views and their relationships, registers the scope of user OLAP queries, and provides information for the subsequent materialization of the views. Then, a hybrid metaheuristic algorithm based on GRASP was discussed as part of an approach that modularizes data structures as well as procedures for views selection and materialization, providing a means of allowing the easy adoption of any other optimization algorithm that receives as input a set of elements with constraints (e.g. possible views with their respective sizes) and a threshold (e.g.

memory space available) and returns a subset of those elements that do not exceed the limit and whose benefit is the maximum value or a value close to the maximum value.

Aiming at assessing the performance and quality of our proposal for VSP based on GRASP and OLAP signatures, performance evaluations were conducted using an extension of the Star Schema Benchmark. The results indicated that GRASP query processing was preferable to a PBS query execution, since the former is a semi-greedy approach that selects a small number of views, representing the profile of user OLAP queries and promoting a reduction in execution time of these queries. In fact, performance tests indicated that compared to PBS, the proposed method obtained a time reduction of about 20.4% in query processing. In addition, GRASP was more scalable than PBS, since it selected and materialized a smaller set of views, even when there was a wide range of possible views to be chosen. To compare GRASP with a recent approach that also selected a small number of views, further experiments were conducted. These aimed at investigating whether GRASP would provide similar reduction in time for selecting views to be materialized and for processing queries over these views as well. GRASP obtained nearly the same query runtime of ACO (i.e. a small performance loss of about 2.84% was obtained by GRASP), but a shorter time for the selection of views than the ACO algorithm (i.e. a gain in processing time of about 77% was produced by GRASP).

The development of a self-configuration mechanism for GRASP input parameters and the complexity analysis of the proposed algorithms are seen as suggestions of future work. Also, the definition of a GRASP dynamic approach to consider costs of updates is another indication of additional research. Finally, it would be interesting to conduct an empirical evaluation to further validate the current version of our conceptual model schema.

## REFERENCES

Alfonseca, E. and Rodríguez, P. Modelling users' interests and needs for an adaptive online information system. In *Proceedings of International Conference on User Modeling.* Pittsburgh, USA, pp. 76–80, 2003.

Aouiche, K., emmanuel Jouve, P., and Darmont, J. Clustering-based materialized view selection in data warehouses. In *Proceedings of Advances in Databases and Information Systems.* Thessaloniki, Greece, pp. 81–95, 2006.

Chan, G., Li, Q., and Feng, L. Optimized design of materialized views in a real-life data warehousing environment. *International Journal of Information Technology* 7 (1): 30–54, 2001.

Chirkova, R., Li, C., and Li, J. Answering queries using materialized views with minimum size. *The Very Large Data Bases Journal* 15 (3): 191–210, 2006.

de Albuquerque Loureiro, J. A. *Restruturação Dinâmica de Estruturas Multidimensionais de Dados em Tempo Útil.* Ph.D. thesis, Universidade do Minho - Escola de Engenharia, Portugal, 2006.

Derakhshan, R., Stantic, B., Korn, O., and Dehne, F. K. H. A. Parallel simulated annealing for materialized view selection in data warehousing environments. In *Proceedings of International Conference on Algorithms and Architectures for Parallel Processing.* Cyprus, pp. 121–132, 2008.

Dobbie, G. and Ling, T. W. Practical approach to selecting data warehouse views using data dependencies. In *Proceedings of International Conference on Conceptual Modeling.* Salt Lake City, UT, USA, pp. 168–182, 2000.

Dorigo, M. and Blum, C. Ant colony optimization theory: a survey. *Theor. Comput. Sci.* 344 (2-3): 243–278, 2005.

Feo, T. A. and Resende, M. G. C. Greedy randomized adaptive search procedures. *Journal of Global Optimization* 6 (2): 109–133, 1995.

Festa, P. and Resende, M. G. C. An annotated bibliography of GRASP – Part I: Algorithms. *International Transactions in Operational Research* 16 (1): 1–24, 2009.

Gou, G., Yu, J., and Lu, H. A/sup */ search: an efficient and flexible approach to materialized view selection. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews* 36 (3): 411–425, 2006.

Hildebrandt, M., Gutwirth, S., Hildebrandt, M., and Gutwirth, S. *Profiling the European Citizen: Cross-Disciplinary Perspectives.* Springer Publishing Company, Incorporated, 2008.

Kalnis, P., Mamoulis, N., and Papadias, D. View selection using randomized search. *Data Knowl. Eng.* 42 (1): 89–111, 2002.

Khan, N. A. and Aziz, A. Partial aggregation for multidimensional online analytical processing structures. *International Journal of Computer and Network Security* 2 (2): 65–71, 2010.

Lawrence, M. Multiobjective genetic algorithms for materialized view selection in olap data warehouses. In *Proceedings of Annual Conference on Genetic and Evolutionary Computation.* Seattle, WA, USA, pp. 699–706, 2006.

Lee, M. and Hammer, J. Speeding up materialized view selection in data warehouses using a randomized algorithm. *International Journal of Cooperative Information Systems* 10 (3): 327, 2001.

Li, X., Qian, X., Jiang, J., and Wang, Z. Shuffled frog leaping algorithm for materialized views selection. In *Proceedings of International Workshop on Education Technology and Computer Science*. Wuhan, China, pp. 7–10, 2010.

Lijuan, Z., Xuebin, G., Linshuang, W., and Qian, S. Research on materialized view selection algorithm in data warehouse. In *Proceedings of International Forum on Computer Science-Technology and Applications*. Chongking, China, pp. 326–329, 2009.

Lin, W.-Y. and Kuo, I.-C. A genetic selection algorithm for olap data cubes. *Knowledge and Information Systems* 6 (1): 83–102, 2004.

Lin, Z., Yang, D., Song, G., and Wang, T. User-oriented materialized view selection. In *Proceedings of IEEE International Conference on Computer and Information Technology*. Aizu-Wakamatsu City, Japan, pp. 133–138, 2007.

Nadeau, T. P. and Teorey, T. J. Achieving Scalability in OLAP Materialized View Selection. In *Proceedings of ACM International Workshop on Data Warehousing and OLAP*. McLean, VA, USA, pp. 28–34, 2002.

OŠNeil, P., OŠNeil, E., Chen, X., and Revilak, S. The star schema benchmark and augmented fact table indexing. In *Performance Evaluation and Benchmarking*, R. Nambiar and M. Poess (Eds.). Lecture Notes in Computer Science. Springer Berlin / Heidelberg, pp. 237–252, 2009.

Phuboon-ob, J. and Auepanwiriyakul, R. Selecting materialized views using two-phase optimization with multiple view processing plan. *International Journal of Computer and Information Engineering* 1 (6): 376–381, 2007.

Serna-Encinas, M. T. and Hoyo-Montano, J. A. Algorithm for selection of materialized views: based on a costs model. *Proceedings of Mexican International Conference on Computer Science* vol. 0, pp. 18–24, 2007.

Shukla, A., Deshpande, P., and Naughton, J. F. Materialized view selection for multidimensional datasets. In *Proceedings of International Conference on Very Large Data Bases*. New York City, USA, pp. 488–499, 1998.

Smith, J. R., Li, C.-S., and Jhingran, A. A Wavelet Framework for Adapting Data Cube Views for OLAP. *IEEE Trans. on Knowl. and Data Eng.* 16 (5): 552–565, 2004.

Song, X. and Gao, L. An ant colony based algorithm for optimal selection of materialized view. In *Proceedings of International Conference on Intelligent Computing and Integrated Systems*. Guilin, China, pp. 534–536, 2010.

Sun, X. and Wang, Z. An efficient materialized views selection algorithm based on pso. In *Proceedings of International Workshop on Intelligent Systems and Applications*. Wuhan, China, pp. 1–4, 2009.

Talebian, S. H. and Kareem, S. A. Using genetic algorithm to select materialized views subject to dual constraints. In *Proceedings of International Conference on Signal Processing Systems*. Singapore, pp. 633–638, 2009.

Theodoratos, D. and Bouzeghoub, M. A general framework for the view selection problem for data warehouse design and evolution. In *Proceedings of ACM international workshop on Data warehousing and OLAP*. McLean, VA, USA, pp. 1–8, 2000.

Vijay Kumar, T. V., Haider, M., and Kumar, S. A view recommendation greedy algorithm for materialized views selection. In *Information Intelligence, Systems, Technology and Management*, S. Dua, S. Sahni, and D. P. Goyal (Eds.). Communications in Computer and Information Science, vol. 141. Springer Berlin Heidelberg, pp. 61–70, 2011.

Wang, Z. and Zhang, D. Optimal genetic view selection algorithm under space constraint. *International Journal of Information Technology* vol. 11, pp. 44–51, 2005.

Whitehorn, M., Zare, R., and Pasumansky, M. *Fast Track to MDX*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

Yang, J., Karlapalem, K., and Li, Q. Algorithms for materialized view design in data warehousing environment. In *Proceedings of International Conference on Very Large Data Bases*. Athens, Greece, pp. 136–145, 1997.

Yu, J. X., Yao, X., Choi, C.-H., and Gou, G. Materialized view selection as constrained evolutionary optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews* 33 (4): 458 –467, 2003.

Yuhang, Z., Qi, L., and Wei, Y. Materialized view selection algorithm-CSSA_VSP. In *Proceedings of International Conference on Computational Intelligence and Natural Computing Proceedings*. Wuhan, China, pp. 68 –71, 2010.

Zhang, C., Yao, X., and Yang, J. An evolutionary approach to materialized views selection in a data warehouse environment. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews* 31 (3): 282 –294, 2001.

Zhang, Q., Sun, X., and Wang, Z. An efficient ma-based materialized views selection algorithm. In *Proceedings of International Conference on Control, Automation and Systems Engineering*. Zhangjiajie, China, pp. 315 –318, 2009.