Reactive GRASP with Path Relinking for Selecting OLAP Views

Andresson da Silva Firmino, Valéria Cesário Times, Ricardo Martins de Abreu Silva
Center for Informatics, Federal University of Pernambuco
50740-560, Recife, PE, Brazil
`{asf2, vct, rmas}@cin.ufpe.br`

Geraldo Robson Mateus
Computer Science Department, Federal University of Minas Gerais
31270-901, Belo Horizonte, MG, Brazil
`mateus@dcc.ufmg.br`

RESUMO

A materialização de visões promove a redução do tempo de execução de consultas multidimensionais. No entanto, esta materialização tem um custo associado que pode ultrapassar as restrições de custo de um determinado ambiente, caracterizando o problema de seleção view (PSV). PSV consiste em selecionar as melhores visões dado um certo limiar de custo. Várias soluções têm sido propostas na literatura para tentar resolver este problema. Neste trabalho, o PSV é tratado por meio de algoritmos de otimização baseados na meta-heurística Reactive GRASP e variantes da heurística Path-Relinking. Os resultados experimentais mostram que o algoritmo proposto é capaz de fornecer melhores soluções quanto o tempo de execução das consultas, em comparação com outras heurísticas na literatura. Nosso algoritmo reativo promoveu uma redução de 10,25 % no tempo de execução das consultas.

PALAVRAS CHAVE. Visões Materializadas. Metaheurísticas. Otimização Combinatória.

Metaheurísticas, Otimização Combinatória, Outras aplicações em PO

ABSTRACT

The materialization of views promotes the reduction of multidimensional queries's execution time. However, this materialization has an associated cost that may exceed the cost constraints of a given environment, characterizing the view selection problem (VSP). VSP consists in selecting the best views given a certain cost threshold. Several solutions have been proposed in the literature to try and solve this problem. In this paper, the VSP is handled by using novel optimization algorithms based on the reactive GRASP meta-heuristic and variants of the path relinking heuristic. The experimental results show that the proposed algorithm is able to provide better solutions regarding the queries runtime, compared to other heuristics in the literature. Our reactive algorithm promoted a reduction of 10.25% in the queries runtime.

KEYWORDS. Materialized Views. Metaheuristic. Combinatorial Optimization.

Metaheuristic, Combinatorial Optimization, Other applications in OP

1. Introduction

In an increasingly dynamic and volatile environment, companies need to identify patterns of behavior and trends present in their business environment, as it is of paramount importance to make immediate decisions in emergent situations in order to remain strong and competitive in the market. Having in mind that a decision taken before competitors might be decisive in market leadership, the need for fast and reliable information encourages the effort to conduct researches on mechanisms to store and manipulate the data appropriately, aiming at speeding up the decision-making process. One may point out the technologies of data warehouse (DW) and OLAP (On-Line Analytic Processing) as a result of these research efforts [Wrembel e Koncilia, 2006].

A DW is a multidimensional database that stores subject-oriented, integrated, time-variant and non-volatile data, and is used to perform analytic queries and information retrieval. In turn, OLAP tools are meant for multidimensional processing of data extracted from DWs, allowing this data to be analyzed from different perspectives and levels of aggregation by decision makers. Multidimensional and analytic queries (OLAP queries) are submitted through OLAP tools, which, in turn, retrieve data from DWs through the execution of operations such as *roll-up*, *drill-down*, *pivoting* and *slice and dice* [Wrembel e Koncilia, 2006]. These operations are applied over a visual metaphor of an structure, called data cube, which organizes the information according to various perspectives of analysis (dimensions, hierarchies and levels) defined by strategic business analysts and for a given analysis fact (information of business interest).

OLAP queries are answered through the execution of operations of selection, join and aggregation over a database structured according to a perspective of analysis (called view). These two last operations are time consuming and use up resources, especially when applied to large volumes of data. However, OLAP queries must be answered in a short time interval, since the quality of decisions, the productivity and the decision makers' satisfaction strongly depend on how fast these queries are processed. For this reason, several solutions have been proposed in the literature to maximize the processing performance of analytic queries. Among these solutions, studies have shown the importance of the partial aggregation of multidimensional structures for speeding the execution of OLAP queries [Khan e Aziz, 2010]. This aspect has raised the interest of the research community, mainly with respect to the identification of which portions of data (i.e. views) must be partially aggregated (i.e. materialized).

Materialized views store aggregated and precomputed data to eliminate the overhead associated with costly join and data aggregation operations required by analytic queries. Thus, given a certain threshold of storage cost, there is a need to select the best views to be materialized, i.e. views that satisfy the storage requirements and provide the lowest response time to process OLAP queries. Several solutions have been proposed by researchers to solve this problem, which is known in the literature as the view selection problem (VSP). Aiming to maximize the performance of OLAP queries, we present in this article the following contributions:

- Proposition of novel optimization algorithms based on the reactive GRASP meta-heuristic and variants of the path relinking heuristic that are used for selecting OLAP views in one of the phases of the proposed method.
- Description of performance tests that validate the proposed method and the proposed optimization algorithms in scenarios with a high diversity of views (i.e. amount of space available for materialization), and runtime of OLAP queries.

This paper is organized as follows: In Section 2, is formally described the VSP considered in this study. Section 3 describes the optimization algorithms that are proposed for selecting views. In Section 4, results collected from performance tests that were conducted in order to evaluate the optimization algorithms are discussed. Finally, Section 5 concludes the article and addresses future work.

2. Problem description

An OLAP systems' user performs a series of analytic queries in order to, for example, analyze the evolution of car sales in various stores located throughout the country. To perform this analysis the user can start by making a query to a level that contains concise data: "*Which stores were the most profitable in the last quarter*?", then obtain more detailed data through the query: "*Which cars were the best sellers in these stores*?"and submit more queries until the analysis is completed. Each submitted query can be answered quickly by views that, when materialized, contain, in their data combination, the data associated with the outcome of the query. However, among views that can answer such query, exist a view that provides the minimum processing cost, i.e., view with smaller size, called view of minimum cost [Firmino et al., 2011]. The execution history of the user's queries indicates what is really relevant to the user at a given time. As also determines the usage frequency for each view of minimum cost. The usage frequency is the view's benefit for processing queries, because views associated with heavily used queries must be materialized to reduce the response time of these queries.

The materialization of all possible views of a data cube is not feasible for the following reasons: First, the materialization of each view has a cost that is affected by several issues, such as storage space and processing time for materializing views. Second, 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. For example, with only two levels and only three dimensions, $2*2*2 = 8$ views are generated, whereas with five dimensions, each with four levels, $4*4*4*4*4 = 1024$ possible views are produced.

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 referred to as view selection problem (VSP) [Khan e Aziz, 2010]. The VSP is a combinatorial optimization problem, which has a set *V* of possible views of a cube *C*, and is essential to find the subset $M \ \in V$ that maximizes the benefit of the total queries performed over *C*.

The VSP mathematical modeling is given by Equation 2, and corresponds to the formulation of the *Knapsack Problem*. Let $V \ = \ \{1, \ \ldots, \ n\}$ be the set of views to be chosen for materialization. Then, x_i is the decision variable that indicates if the view i was chosen to be materialized, assuming the value of 1 if it was indeed chosen and of 0 otherwise. Let B_i be the benefit of materialization for the view i and S_i the physical space required for the storage of the view i. Then, the VSP problem is to find a subset of V that allows to obtain the greatest benefit value considering CV as a constraint value, which represents the storage space available for the materialization of the views selected.

$$Max\ f\,(x) = \left\{ \sum_{i=1}^{n} x_i{*}B_i \right\}$$

$$Subject\ to: \sum_{i=1}^{n} x_i{*}S_i \leq CV and\ x_i \in \{0,1\}\,, \forall\, i = 1, \ldots, n \tag{1}$$

3. Using GRASP to Select OLAP Views

Two optimization algorithms proposed are based on the following meta-heuristics: Reactive GRASP(RG), described in Section 3.1 , and Reactive GRASP with Path Relinking (RGPR), which uses the Path Relinking heuristic and is described in Section 3.2. These algorithms are detailed as follows. These optimization algorithms use the novel local search technique described in Section 3.3.

3.1. Reactive GRASP (RG)

The RG algoritm, described by Algorithm 1, extends our previous work for selecting OLAP views [Firmino et al., 2011] as follows: First, the greedy rate, which was obtained previously by receiving it as an input parameter, is now computed by the procedure *GetGreedyRate* that reactively calculates its value. Finally, RG uses a new heuristic in the phase of local search that eliminates the use of two input parameters (ϑ and τ) present in the method proposed in [Firmino et al., 2011].

Algorithm 1 Reactive GRASP Algorithm

```
Require: $\varphi$, a list of views
Require: $\delta$, amount of space available for materialization
Ensure: best found solution
1: $bestsolution \leftarrow \emptyset$
2: while stopping criteria not satisfied do
3:     $\varphi' \leftarrow \varphi$
4:     $solution \leftarrow$ ConstructionPhase($\varphi'$,$GetGreedyRate()$,$\delta$)
5:     $solution \leftarrow$ LocalSearchPhase($\varphi'$, $\delta$, $solution$)
6:     UpdateBestSolution($solution$, $bestsolution$)
7: end while
8: return $bestsolution$
```

At the beginning of the algorithm 1, the best solution is initialized to an empty set of views, i.e. at the beginning of the algorithm, the benefit of the best solution is zero. In lines 2 to 7, the GRASP iterations are performed until the stop criterion is met. In line 3, the variable φ is copied into φ in the beginning of each new iteration, so that the list of views given as input to the procedure *ConstructionPhase* of line 4 remains unchanged. In line 4, a partial solution is obtained through the construction phase and is stored in the variable *solution*. Afterwards, in line 5, during the phase of local search, a new solution, which is equal to or better than the previous solution, is obtained and stored in the variable *solution*. In line 6, the procedure *UpdateBestSolution* is called to compare the current best solution (*bestSolution*) and the solution obtained in the previous line (*solution*). It subsequently assigns the solution with greatest benefit to *bestSolution*. The benefit of a solution is given by the sum of the benefit of each view that composes the solution. Finally, at the end of all iterations, in line 7, the best solution found so far, which is essentially the best set of views for processing queries of a given signature, is returned.

The procedure *GetGreedyRate* works as follows. Initially, an initial value between {0,1} is assigned to the greedy rate θ. Then, an initial change that can be {increment (I) or decrement (D)} and a value *v* to be incremented or decremented are defined. At each algorithm iteration, there is a check to assess whether the current solution has improved in relation to the previous one, i.e. if the current generated solution is better than the previous solution. If it is, the current change is performed. Otherwise, the opposite change is conducted.

3.2. Reactive GRASP with Path Relinking (RGPR)

The RGPR algoritm, described by Algorithm 2, is a result of the combination of RG with Path Reliking. Path Relinking (PR) is used by this algorithm within the GRASP metaheuristic's iterations, as suggested in [Mateus et al., 2011]. The purpose of this Path Relinking application is to run Path Relinking with different, good elite sets, because during the algorithm iterations, the elite set is adjusted so as to improve the quality of re-linkings.

At the start of Algorithm 2, the elite set £ is initialized with an empty set. The iterations of this algorithm are computed from lines 2 to 10 until the stop criterion is reached. During each iteration, a solution resulting from the local search is generated in line 5. In line 6, the algorithm checks if the elite set £ is full. If £ is full, in line 7, the path relinking is applied, and then, to add the generated solution to £, the method *ADD* is called in line 8. It is worth mentioning that the solution is only added if the difference between itself and all the solutions contained in £ is greater than the value returned by the procedure *GetDifferenceThreshold*. This constraint is important to preserve the diversity of the elite set's solutions. The value obtained reactively by *GetDifferenceThreshold* represents the threshold of the difference between two solutions, i.e., the minimum number of different elements between two solutions so as to consider the two solutions sufficiently distinct.

Otherwise, if £ is not full yet, then the method *ADD* adds the solution to £ in line 10. Finally, at the end of the iteration, in line 11, the best solution found in the elite set is returned.

GetDifferenceThreshold works as follows. Initially, the elite set £ must have at least two solutions. The difference between them is computed and considered the current difference threshold (Δ). When a new solution is added to the elite set £, the value of Δ is updated using the following equation: $\Delta{=}\text{Max}(\Delta, \text{Diff}(\text{sol}_{\text{best}}, \text{sol}_{\text{best}-1}))$. sol_{best} and $\text{sol}_{\text{best}-1}$ represent respectively the best solution and the second best solution for £.

Algorithm 2 Reactive GRASP with PR Algorithm

```
Require: $\varphi$, a list of views
Require: $\delta$, amount of space available for materialization
Require: $\xi$, path relinking variant
Ensure: best found solution
 1: £ ← $\emptyset$
 2: while stopping criteria not satisfied do
 3:     $\varphi' \leftarrow \varphi$
 4:     $solution$ ← ConstructionPhase($\varphi'$,$GetGreedyRate()$,$\delta$)
 5:     $solution$ ← LocalSearchPhase($\varphi'$, $\delta$, $solution$)
 6:     if IsFull(£) then
 7:         $solution$ ← PathRelinking(£,$solution$,$\xi$)
 8:         ADD(£, $solution$, GetDifferenceThreshold())
 9:     else
10:         ADD(£, $solution$, GetDifferenceThreshold())
11:     end if
12: end while
13: return MAX(£)
```

The *PathRelinking* procedure consists consists in a search strategy of solutions that exploits the connection path between two solutions: the $solution$ and the a solution is chosen randomly from £. The process of connection path is performed until the truncation condition is reached or both solutions are identical. The truncation condition is true, while the percentage of improvements in the solutions generated by PR is greater than the percentage of no improvements. The connection step can be performed, according to the relinking variant (ξ): *forward*, *forward truncated*, *backward*, *backward truncate*, *mixed* (i.e. forward and backward are executed interchangeably) and *mixed truncate*. The connection steps may be made by adding or removing views. The choice of the next connection step to be executed is performed by computing all possible movements and selecting the movement with greater benefits. At the end of the procedure, the best solution found during relinking process is returned.

3.3. The Novel Local Search

In each iteration of the GRASP meta-heuristic, the procedures *ConstructionPhase* and *LocalSearchPhase* are called. The *ConstructionPhase* procedure aims at generating a good initial solution by means of a semi-greedy heuristic. The *ConstructionPhase* procedure, used by the proposed algorithms, is the same developed in [Firmino et al., 2011]. The second procedure aims to refine the solution obtained in the construction phase in order to find better solutions. Given a solution s, the search for better solutions is made through the search for neighboring solutions in order to find better solutions than s

In [Firmino et al., 2011], the local search technique is based on the random choice of elements of the exchange set, so that a valid permutation is performed. However, there is a low probability of valid permutation because of the random choice of elements. Differently from the local search technique used in [Firmino et al., 2011], in this article, we propose a novel local search technique, described in Algorithm 3, to overcome the low efficiency of the swap method applied in the aforementioned work. The novel technique consists in sorting the views of φ and *solution* and in finding permutations in the sorted lists of views. Based on their benefit values, the views are ordered in descending order and in case of a tie, they are ordered in an increasing manner and based on their sizes, as shown in line 1. Due to the ordering of these elements, it is possible to know in advance that if the current element cannot be exchanged, then it will not be possible to exchange its subsequent elements.

Algorithm 3 LocalSearchPhase Procedure

```
Require: $\varphi$, a list of views
Require: $\delta$, amount of space available for materialization
Require: $solution$, solution built in the previous phase
Ensure: a similar or improved solution
 1: SortByValueDescAndSizeAsc($solution$,$\varphi$)
 2: SetAll($selectedList$,false)
 3: $p^{sol} \leftarrow$ Length($solution$) - 1; $p^{\varphi} \leftarrow 0$
 4: $isStopping \leftarrow$ false
 5: $swapList \leftarrow \emptyset$
 6: while not $isStopping$ do
 7:     $p^{\varphi} \leftarrow$ GetNextNotSelect($selectedList$,$p^{\varphi}$)
 8:     $ElmIn \leftarrow$ Get($solution$,$p^{sol}$)
 9:     $ElmOut \leftarrow$ Get($\varphi$,$p^{\varphi}$)
10:     if Value($ElmIn$) = Value($ElmOut$) then
11:         if Size($ElmIn$) > Size($ElmOut$) then
12:             RecordSwap($selectedList$, $swapList$, $p^{sol}$, $p^{\varphi}$)
13:         else
14:             $isStopping \leftarrow$ true
15:         end if
16:     else if Value($ElmIn$) > Value($ElmOut$) then
17:         $isStopping \leftarrow$ true
18:     else if SizePlus($ElmIn$) > Size($ElmOut$) then
19:         RecordSwap($selectedList$, $swapList$, $p^{sol}$, $p^{\varphi}$)
20:     else
21:         SearchCombination($p^{sol}$, $p^{\varphi}$)
22:     end if
23:     if $p^{\varphi}$ > Length($\varphi$) then
24:         $p^{sol} \leftarrow p^{sol}$ - 1; $p^{\varphi} \leftarrow 0$
25:     end if
26:     if $p^{sol} < 0$ then
27:         $isStopping \leftarrow$ true
28:     end if
29: end while
30: return ExecuteSwaps($swapList$,$\varphi$,$solution$)
```

In second line of Algorithm 3, a list of boolean values whose length is equal to the sum of the length of the two lists (φ and *solution*) is initialized with all values equal to false. This list indicates the elements that were selected during the execution for being exchanged (*selectedList*). In line 3, two pointers (i.e. p^{sol} and p^{φ}) are initialized. The pointer p^{sol} is used to scan the ordered list of views in *Solution* from the end to the top of this list, while p^{φ} is used to scan the ordered list of views in φ from the top to the end of this list. In lines 4 and 5, the iterations' control variable (i.e. *isStopping*) and the list used to store the exchanges made during the iterations (i.e. *swapList*) are initialized, respectively.

When all variables are initialized, the algorithm's iterations are executed in lines 6 to 29. At the beginning of the iteration, the pointer is updated to point to the next view in φ that has not been selected for exchanging. Through the pointers and the procedure *Get*, the view that is signaled by the pointer is obtained. The procedures *Value* and *Size* receive as input parameter a view, returning respectively the value and the size of this view. The procedure *SizePlus* returns the view's size plus the remaining space in *solution*. The remaining space represents the space available in *solution* to receive other views so that the size of *solution* does not exceed the space available for materialization. The remaining space is calculated by subtracting the size of *solution* from the total space available for materialization (δ). The size of *solution* is calculated through a sum of the sizes of all views contained in *solution*.

In line 10, there is a check to assess if the view inside the solution (*ElmIn*) has the same value as the outside view (*ElmOut*). If that happens, there is a second check (line 11) to assess if the size of *ElmIn* is larger than *ElmOut*. If this is established to be true, the exchange is recorded (as shown line 12) because this frees space in *solution* without affecting the value of *solution*. However, if the size of *ElmIn* is not greater than *ElmOut*, the iterations are terminated (line 14). This occurs, because, since the elements are ordered, the next elements to be scanned in φ have sizes that are equal to or greater than *ElmOut*, preventing the permutation with any element of φ. When the value of *ElmIn* is greater than *ElmOut*, in line 17, the iterations are also terminated. This occurs because of the ordering of the views, which indicates that the next views to be scanned in φ have values that are equal to or smaller than *ElmOut*. Thus, this prevents any permutation with *ElmIn* or any element of *solution*. In line 19, if the value of *ElmIn* is smaller than *ElmOut* and the size of *ElmIn* plus the remaining space of *solution* is greater than the size of *ElmExt*, the exchange is stored.

To investigate cases where views could be exchanged, comparisons based on the values and sizes of the views of *ElmIn* and *ElmOut* were made. However, an untreated case remains, when *ElmIn* is smaller than *ElmOut* and the size of *ElmIn* plus the remaining space of *solution* is smaller than the size of *ElmOut*, i.e. *ElmOut* is too large to be exchanged. In this case, in line 21, the procedure *SearchCombination* is executed. Using the sorted list, this procedure searches for the closest views to *ElmIn*, so as to find a combination of views whose combined size is greater than *ElmOut* and whose combined value is smaller than *ElmOut*. Having found this combination, the procedure *SearchCombination* calls the procedure *RecordSwap* to register the exchange. The procedure *RecordSwap* stores in *swapList* the exchange to be made and adjusts the pointers to continue the searches for new permutations. The adjustment of pointers is done in line 24.

At the end of each iteration of the Algorithm 3, pointers are checked. In line 24, the pointers are adjusted if the pointer p^{φ} has scanned all elements in φ and no permutation compatible with the current *ElmIn* has been found. In addition, in line 27, if all elements of *solution* have been scanned, i.e. $p^{sol} < 0$, then the iterations are terminated. Finally, the procedure *ExecuteSwaps* performs all the exchanges recorded in the *swapList* and returns the solution resulting from the exchanges that were carried out.

4. Performance Evaluation

4.1. Experimental Setup

Experiments were performed on a computer with the *Windows 7 Professional 64 bits* operating system, *Intel Core i7 X 980 @ 3.33GHz* processor and 16GB of RAM. The OLAP server used was the *Mondrian* version 3.2.0.13583. *Mondrian* is an open source OLAP server, written in *Java*, which runs queries written in MDX, by reading data from a relational database and presenting the results in a multidimensional format. All of the algorithms have been implemented in *Java*. The generator of randomness was the *Mersenne Twister* found in the *COLT1* library[1]. The seeds were obtained from decimal places of $\pi = .14159265358979323 84...$ taken from 5 to 5, e.g. the $seed_1$ = 14159, $seed_2$ = 26535, $seed_3$ = 89793, and so on.

The *Star Schema Benchmark* (SSB)[ONeil et al., 2009] was adapted to increase the number of dimensions and levels, in order to have a large number of views and, consequently, a greater number of different queries. This change was necessary to investigate the scalability and efficiency of the proposed algorithms when there is a large set of possible views to be chosen. With respect to the schema of multidimensional data used, it is composed of 5 dimensions (*Part*, *Supplier*, *Customer*, *Date* and *CommitDate*), 37 levels and 2 measures (*Lo_Revenue* and *Lo_SupplyCost*). This schema was used to create a synthetic dataset based on the SSB's scale factor 1, which produced

[1]Available at *http://acs.lbl.gov/software/colt/colt-download/releases*

about 6M records in the fact table and produced a total of 17,325 possible views, whose materialization would consume the estimated space of 8,346 TB.

The workload was composed of 1,200,006 query submissions randomly chosen from 1200 distinct queries. These 1200 queries were chosen at random, with seed 1 belonging to a set *k* of 17,325 distinct queries. Each query *k* was designed to have a unique combination of levels, and therefore, each of them is directly associated with only one view of the 17,325 possible views. The estimated total space for the materialization of 1,200 views associated with the 1,200 user's distinct queries is 561.2 GB. Given this total amount of space, in the experiments the following percentages of space available for materialization were considered: 1.25% = 7GB; 2.5% = 14GB; 5% = 28GB; 10% = 56GB; 20% = 112GB; 40% = 224GB and 60% = 336GB.

In our experiments, nine algorithms were evaluated: ACO [Song e Gao, 2010], G (Firmino, et al. , 2011), and seven instances of the novel algorithms proposed in this article: RG (Reactive GRASP as described by Algorithm 1), RGF (RG with relinking of type *forward*), RGFT (RGF with the truncation condition), RGB (RG with relinking of type *backward*), RGBT (RGB with the truncation condition), RGM (RG with the mixed relinking, i.e. forward and backward) and RGMT (RGM with the truncation condition). The reason for using an ACO (*Ant Colony Optimization*) algorithm in our study is twofold. First, has been applied to several optimization problems [Dorigo e Blum, 2005]. Second, ACO was evaluated with a small workload of 32 distinct queries, and presented a somewhat better performance than the experiments reported in [Firmino et al., 2011]. Then, in this paper, we aim to evaluate the ACO with respect to our GRASP-based algorithms by using a larger workload composed of 1200 distinct queries.

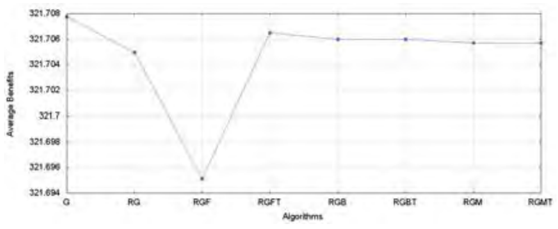
For all the algorithms no-reactive, i.e. ACO and G, it was necessary to perform the parameters' calibration for the execution of the experiments detailed in this article. The calibration process was same used in [Firmino et al., 2011].

4.2. Performance Results

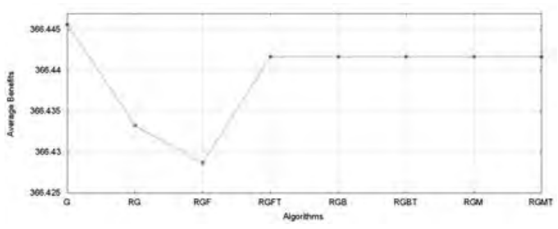
In this section, a comparative analysis between the nine algorithms is performed. These are ACO [Song e Gao, 2010], G [Firmino et al., 2011], RG, RGF, RGFT, RGB, RGBT, RGM and RGMT. This analysis was carried out using two evaluation criteria: (1) space available for materialization and (2) runtime reduction of the user's OLAP queries. These criteria defined two test configurations used in our experiments that are described as follows.

Space Available for Materialization. The aim of this test configuration is to evaluate the behavior of the algorithms in different scenarios of available space for materialization: {1,25%, 2,5%, 5%, 10%, 20%, 40% e 60%}. To accomplish that, each algorithm was executed 50 times, using as a stop criterion the maximum time of 2 minutes and the value of 500000 as the maximum number of iterations of the algorithm without improvements in the quality of the solution generated. We also used as seed the number corresponding to the execution number. For the non-reactive algorithms (ACO and G), we used their respective values of parameters that were obtained from the calibration process. We calculated the average of the generated solutions' benefits for all 50 executions of each algorithm in the different scenarios of available space. This is shown in Figure 1. It is important to notice that the ACO's benefit values (i.e. 1,25% = 273,20; 2,5% = 311,65; 5% = 337,94; 10% = 409,21; 20% = 513,35; 40% = 683,92 e 60% = 834,62) are not shown in these figures because they were much smaller in value than the results of the other algorithms. For that reason, their inclusion would prevent a more detailed visualization of all the results, being therefore suppressed in the image.

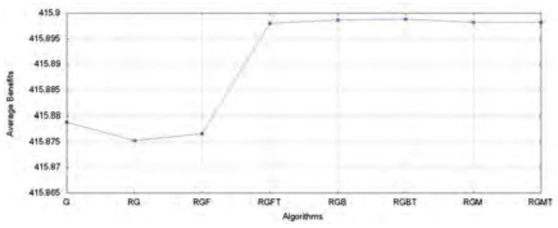
The PR's truncated variants produced performance gains with respect to their corresponding non-truncated approaches, as shown especially in the spaces 20% and 40%. This occurred because in certain scenarios there is no need to execute the entire path of relinking if better solutions can be found in the path's intermediate parts. Hence, this processing gain can be used to generate new solutions in the construction phase or in the local search phase of the GRASP meta-heuristic.



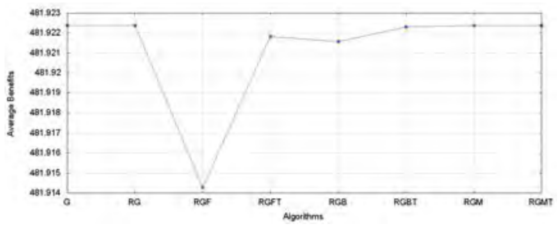
(a) Space 1.25%



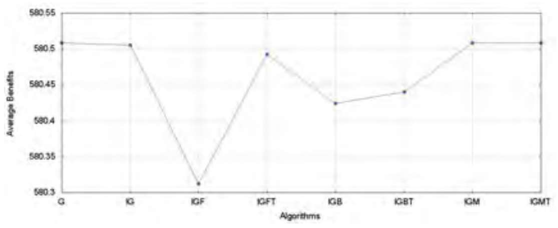
(b) Space 2.5%



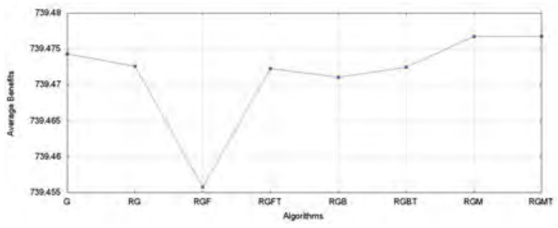
(c) Space 5%



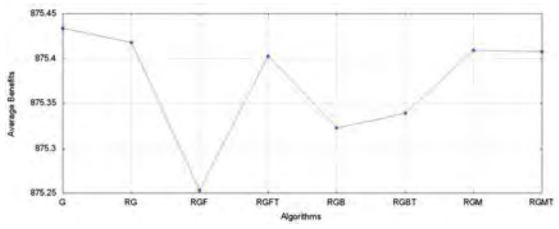
(d) Space 10%



(e) Space 20%



(f) Space 40%



(g) Space 60%

Figura 1: Assessment of space available for materialization

Also, it is important to notice that in most cases, the RGF algorithm (i.e. Reactive GRASP with relinking of type *forward*) presented performance losses when compared to other algorithms. This occurred because RGF is a non-truncated variant, as explained previously, and because in certain scenarios, a permanently ascending search in the space of solutions (i.e. from a worst solution to a better solution) may not be beneficial if between the two ends one can only find worse solutions than those in the extremes.

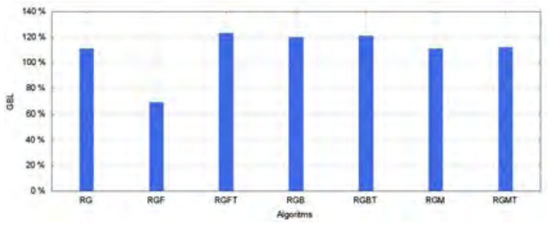
In addition, the G algorithm, generated slightly better results on average than its reactive variant, called RG. This occurred because the G algorithm went through a calibration phase, being therefore expected since the beginning to produce good solutions thanks to its calibration. On the other hand, due to its reactive properties, the RG algorithm takes some time until its parameters are stabilized, and during this time interval, it might generate not so good solutions, reducing the benefit average value of the RG's solutions. One should note that the addition of the Path Relinking meta-heuristic to the RG algorithm produced, in most cases, an increase in the quality of the solutions generated. Then, the average value of all solutions generated for all the spaces of materialization were evaluated, and we noticed that the reactive algorithms presented slight losses when compared to the G algorithm. The greatest performance loss, in a reactive algorithm, was of 0.01152%.

In this work, in addition to the averages of the solutions generated, we are also interested in analyzing the gains obtained by our novel technique of local search, which is proposed in this paper. Thus, for the 50 executions related to the spatial evaluation, we calculated the average gains of this local search by using Equation 2.

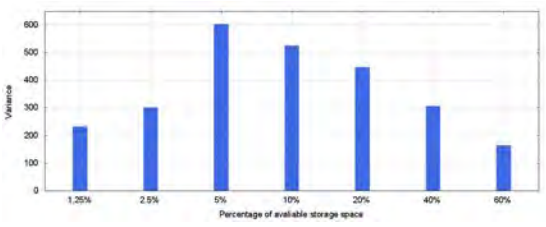
$$\frac{\sum_{i=1}^{i=N} solC_i - solM_i}{N}, where : \qquad (2)$$

- *i* represents the index of the algorithm's ith-iteration;
- *solC* indicates the value of the solution produced in the construction phase;
- *solM* corresponds to the value of the solution produced in the local search phase.

The local search gains represents the gains promoted by the local search technique used in the executed algorithm. In this work, we compared G and the reactive algorithms with the novel local search technique. Results indicated that all reactive algorithms generated performance gains when compared to G's local search. The lowest percentage gain was 69%, as shown in Figure 2.



(a) Percentual Gains of the Novel Local Search



(b) Variance vs. Space available for Materialization

Figura 2: Assessment of Novel Local Search and Variance

We also investigated the behavior of the solution space, by allocating different values to the space available for materialization. In the 50 executions made for this evaluation, we calculated the variance of the solutions generated by the algorithms for the different spaces available for materialization. The idea was to investigate the diversity of the solution space because, in environments of greater diversity of solutions, there is a greater incidence and amplitude of local maxima in the solution space. This irregularity in the solution space represents an extra challenge for the algorithms while they are searching for excellent or very good solutions, since they have a higher probability of becoming trapped in local optima.

As shown in Figure 2, for small or large spaces of materialization, the variances obtained were smaller than those obtained with intermediate sizes of spaces for materialization. This occurred because, with small spaces, the number of views that can be added to the solution is small, reducing the problem's difficulty, since only a few views need to be considered to be chosen and consequently, few different solutions are generated. On the other hand, with large spaces of materialization, many views can be added to the solution, due to the large space available. Initially, a large number of views is added to the solutions because there is a large space for materialization. Then, at the end, there is only a small number of views to be chosen because most of them were already selected and consequently, only few different solutions are generated. This determines the solutions' diversity and thus, the irregularity of the solution space. Thus, we verified that the scenario of greater diversity and, consequently, of greater difficulty, was the space available for materialization equal to 5%, which presented the largest variance.

Runtime Reduction of OLAP Queries. In this test configuration, the views selected by the algorithms were materialized and the time spent to process the history of user queries was compared, using for each algorithm its respective solution (i.e.views selected by each algorithm). After that step, we selected the best solution produced by reactive algorithms (called *SReactiveGRASP*) and the best solution produced by the ACO algorithm (called *SACO*). The algorithms selected the views using the percentage value of 5% of the total space for materialization. The value of 5% was used because it corresponds to the scenario of greatest complexity, as detailed previously.

For each solution, we carried out 20 executions of each query q of the 1200 MDX distinct queries of the history of the user's queries. Then, we collected the elapsed runtime and computed the average runtime for the 20 executions of each query q. Multiplying the average runtime of the query q by the respective frequency of the query q found in the history of the user's queries, we obtained the query q's runtime with respect to this history of queries. Finally, we computed the sum of the runtime of each query q found in the history of queries, and collected the runtime needed to solve the history of the user's queries using the aforementioned solution.

To compare the average runtime of two solutions, we performed the *Paired T Test* [Nancy L. Leech, 2007], which has been used to compare the means of two samples, so that the observations of a sample are paired with observations of another sample. Through two samples, X and Y, we created a set D with the differences between the measurements of each sample $(d_i = y_i - x_i)$. However, to apply the paired T-test, the differences between the averages must have an approximately normal distribution. Then, the *Jarque-Bera Test for Normality* [Jarque et al., 1987] was applied to the 1200 MDX distinct queries to ascertain if the obtained samples follow a normal distribution. After having satisfied the normal distribution constraint, we built a confidence interval for the difference of the averages and checked if the two samples were equivalent statistically. Then, for each pair of samples of the two solutions being compared, the Paired T-Test with a confidence interval of 99% was computed. The idea is to verify if the average runtime of a query q is statistically the same for both solutions. If this is true, the average runtime of the query q using both solutions is the same.

According to our *Paired T Test* results, in 74% (890 of 1,200) of the queries, we concluded that the average runtime was different for both solutions. Also, for each solution (*SReactiveGRASP* and *SACO*), we computed the sum of the runtime of each query from the set of 1200 MDX distinct queries, in order to get the runtime needed to solve the history of the user's queries. Results indicated that the *SReactiveGRASP* solution produced a runtime reduction of 10.25% when compared to the *SACO* solution.

5. Conclusion and Future Work

We proposed optimization algorithms based on the Reactive GRASP meta-heuristic and Reactive GRASP with Path Relinking, for view selection problem (VSP), aiming at maximizing the performance of OLAP queries. To the best of our knowledge, our work makes the first attempt of applying the GRASP meta-heuristic with Path Relinking and its variants to VSP. Furthermore, was proposed a novel local search technique which is used by our proposed optimization algorithms.

Experiments were performed using two test configurations: Space Available for Materialization and Runtime Reduction of OLAP Queries. In the first test configuration, we aimed at studying: (1) the behavior of the solution space when there are variations in values of space available for materialization; (2) the local search improvements, which stand for the gains promoted by the local search technique of the tested algorithms; (3) the usage of the Path Relinking heuristic; (4) the non-reactive and reactive properties of the algorithms tested. The novel local search of our reactive algorithms produced performance gains of at least 69% when compared to previous local search. The second test configuration evaluated the reduction in the response time of queries executed over the user's history, applying the views of two solutions generated by two algorithms: Reactive GRASP and ACO. Results showed that the Reactive GRASP's solution produced a runtime reduction of 10.25% when compared to the ACO's solution.

The definition of multi-objective optimization algorithms for VSP problem is a possibility for future research. That is, new criteria, besides the views' usage frequency, need to be studied and treated together. Examples are maintenance cost and the use of indexes in tables associated at materialized views. Finally, we plan to investigate how materialization tables can be distributed in a distributed environment to optimize queries runtime.

Referências

Dorigo, M. e Blum, C. (2005). Ant colony optimization theory: a survey. *Theoretical Computer Science*, 344:243–278.

Firmino, A., Mateus, R. C., Times, V. C., Cabral, L. F., Siqueira, T. L. L., Ciferri, R. R., e de Aguiar Ciferri, C. D. (2011). A novel method for selecting and materializing views based on olap signatures and grasp. *Journal of Information and Data Management*, 2(3):479–494.

Jarque, M., Bera, A. K., Jarque, C. M., e Bera, A. K. (1987). A test for normality of observations and regression residuals. *International Statistical Review*, 55:163–172.

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

Mateus, G. R., Resende, M. G., e Silva, R. M. (2011). Grasp with path-relinking for the generalized quadratic assignment problem. *Journal of Heuristics*, 17(5):527–565.

Nancy L. Leech, L. D., Anthony J. Onwuegbuzie (2007). Paired samples t test (dependent samples t test). In *Encyclopedia of Measurement and Statistics*, p. 724–727. SAGE Publications, Inc., 0 edition.

ONeil, P., ONeil, E., Chen, X., e Revilak, S. (2009). The star schema benchmark and augmented fact table indexing. In *Performance Evaluation and Benchmarking*, Lecture Notes in Computer Science, p. 237–252. Springer Berlin / Heidelberg.

Song, X. e Gao, L. (2010). An ant colony based algorithm for optimal selection of materialized view. In *Proc. Int Intelligent Computing and Integrated Systems (ICISS) Conf*, p. 534–536.

Wrembel, R. e Koncilia, C. (2006). *Data Warehouse and OLAP: Concepts, Architectures, and Solutions*. IRM Press.