A GRASP Heuristic to Optimize the Materialization of Views in the Cloud

Vilmar Jefté Rodrigues de Sousa, Michael David de Souza Dutra, Bruno Bachelet, Laurent d'Orazio

A GRASP HEURISTIC TO OPTIMIZE THE MATERIALIZATION OF VIEWS IN THE CLOUD

Vilmar Jefté Rodrigues de Sousa
Universidade Federal de Minas Gerais, UFMG, Engenharia de Produção.
Belo Horizonte - Minas Gerais, Brasil
vilmarjet@yahoo.com.br

Michael David de Souza Dutra
Universidade Federal de Minas Gerais, UFMG, Engenharia de Produção.
Belo Horizonte - Minas Gerais, Brasil
michaeldavid@ufmg.com.br

Bruno Bachelet
Clermont Université, CNRS, Université Blaise Pascal, LIMOS UMR 6158
Clermont-Ferrand, France
bruno.bachelet@univ-bpclermont.fr

Laurent d'Orazio
Clermont Université, CNRS, Université Blaise Pascal, LIMOS UMR 6158
Clermont-Ferrand, France
laurent.dorazio@univ-bpclermont.fr

ABSTRACT

This paper studies the problem of materializing views for a database stored in the cloud, in order to improve the response time of queries on the database under a budget limit. Views are subsets of the database, also stored in the cloud, that act like caches to accelerate the access to data. In a cloud, CPU usage and data storage have to be paid, meaning that a trade-off between speed gain and storage cost has to be found. The problem has been formulated as a mixed integer program (MIP) and identified as NP-hard. To solve the problem a GRASP heuristic is proposed. The quality and speed of this algorithm is compared with the results of a MIP solver. Numerical experiments on many randomly generated instances of the problem show that the proposed approach is able to efficiently solve large instances of the problem.

KEYWORDS. GRASP metaheuristic, mixed integer programming, view materialization, cloud pricing.

Main area: Metaheuristica

1. Introduction

Cloud computing has recently emerged as a new way of using software and hardware resources. Instead of setting up and maintaining his own infrastructure, a user can rent resources from a provider. The user only pays for the resources he uses, meaning for transferring data from and into the cloud, and for storing data and processing in the cloud. Hence, for economic reasons, optimizing resources usage in the cloud is essential.

One way of improving the performance of a database, and thus reducing its operating cost in a cloud, is to materialize views. A view is a subset of the database, and more precisely, it is the result of a given query on the database. To materialize a view is to run the query that will produce the view and to store the result. Then, assuming that a query needs some data stored in a view, the processing time of this query can be reduced by accessing directly the data inside the view instead of getting these data by querying the whole database.

Therefore, materializing a view can improve the response time of a query, but at the expense of creating this view (processing and storing) and maintaining it (an update in the database may induce an update of the view). Therefore, the decision of materializing a view is a trade-off between improving the response time of queries and reducing the operating cost in the cloud. In this study, we consider a database where queries $Q = \{Q_i\}_{i=1..m}$ are known for a given period of time. The problem of selecting the views to be materialized in order to minimize the time for processing all the queries in Q has been addressed in Yang (1997), but it was not in the context of cloud computing, meaning that no charge for storing and processing were considered.

In Nguyen (2012), a model of pricing in a cloud has been proposed, and based on this model, a mixed integer program (MIP) has been introduced in Perriot (2013) to formulate the problem of materializing views for a database stored in a cloud to optimize the processing time under a budget limit. In this work, a set of candidate views $V = \{V_k\}_{k=1..p}$ is given (it will be provided by a view selection algorithm, or by experts), and the response times with and without views are obtained by experiments on an actual cloud. This problem has been solved with the MIP solver CPLEX[1], which, for large instances, could not provide an optimal solution in a limited time.

In this paper, we propose a GRASP heuristic to solve the problem. The remainder of this paper is organized as follows. In Section 2, we introduce the background information and the notations used throughout the paper, notably concerning the pricing model and the optimization problem. In Section 3, the GRASP metaheuristic and its implementation to solve the discussed problem are presented. In Section 4, we compare experimental results of our algorithm with those obtained by the MIP solver CPLEX, for randomly generated instances of various sizes (i.e., various numbers of views and queries). Finally, in Section 5, we conclude this article.

2. Background

Let $Q = \{Q_i\}_{i=1..m}$ be the set of queries to be processed on the database, and $V = \{V_k\}_{k=1..p}$ the set of candidate views assumed to be preselected by an existing algorithm, such as in Baril (2003). The problem is to decide which views among the candidates must be materialized. For this purpose, decision variables x_k are introduced: $x_k = 1$ if view V_k is selected for materialization, and $x_k = 0$ otherwise.

2.1. Cloud Pricing Policies

Cloud Service Providers (CSPs), such as Microsoft Azure and Amazon Elastic Computing Cloud (EC2), charge the users for three different resources: the storage cost (C_s), the data transfer cost (C_t), and the computing cost (C_c). Thus, the total cost (C_{total}) is:

$$C_{total} = C_s + C_t + C_c \qquad (1)$$

[1] IBM ILOG CPLEX Optimizer: http://www.ibm.com/software/integration/optimization/cplex-optimizer

2.2. Storage Cost (C_s)

The user has to pay for all data stored in the cloud over the operating period. Many pricing policies exist, but we consider here a fixed price c_s for each GB of stored data per month. Let S_D be the size of the whole database, s_k the size of view V_k, and T the processing period length (selected views are assumed to be materialized during the whole period). Therefore, the user has to pay for the storage of the database and the materialized views. The storage cost is:

$$C_s = c_s \times T \times \left(S_D + \sum_{k=1}^{p} s_k \times x_k \right) \quad (2)$$

2.3. Data Transfer Cost (C_t)

The user has to pay for all the data transferred from and into the cloud. Usually, CSP offers the upload transfer fees, so we ignore them here. Only the download transfer is considered, and in our case, it is only the reception of the answers to queries. Let S_A be the size of the answers to all the queries in Q, and c_t a fixed price for each GB of downloaded data. The data transfer cost is:

$$C_t = c_t \times S_A \quad (3)$$

2.4. Computing Cost (C_c)

The user has to pay for all the processing time spent in the cloud. It is the time needed to process the queries (noted T_{proc}), but also the time to materialize and to maintain the views. Let T_k be the time needed to materialize and maintain view V_k, and c_c a fixed price for each hour of CPU processing. The computing cost is:

$$C_c = c_c \times \left(T_{proc} + \sum_{k=1}^{p} T_k \times x_k \right) \quad (4)$$

Note that the time to process a query depends on whether this query uses a view or not. Let T_i be the processing time of query Q_i without any view, and g_{ik} be the gain on processing time of using view V_k for processing query Q_i. If view V_k brings no gain to query Q_i, then $g_{ik} = 0$. The gains g_{ik} are hard to estimate, and determining analytical expressions of those gains for a cloud is an open issue; therefore these estimations are obtained through experiments on an actual cloud, Perriot (2003). We assume that a query can use at most one view, and such use is represented by decision variables x_{ik}: $x_{ik} = 1$ if query Q_i uses materialized view V_k, and $x_{ik} = 0$ otherwise. The time for processing all the queries of Q is:

$$T_{proc} = \sum_{i=1}^{m} \left(T_i - \sum_{k=1}^{p} g_{ik} \times x_{ik} \right) \quad (5)$$

2.5. Mixed Integer Program

In this paper, the problem is to minimize the processing time T_{proc} under a budget limit $C_{total} \leq C_{max}$. Based on the equations *(1)* to *(5)*, the optimization problem is expressed as a mixed integer program:

$$
(P)\begin{cases}
\min & T_{proc} \\
\\
s.t. & C_{total} = C_s + C_t + C_c \le C_{\max} & (1)\\
& C_s = c_s \times T \times \left(S_D + \sum_{k=1}^{p} s_k \times x_k \right) & (2)\\
& C_t = c_t \times S_A & (3)\\
& C_c = c_c \times \left(T_{proc} + \sum_{k=1}^{p} T_k \times x_k \right) & (4)\\
& T_{proc} = \sum_{i=1}^{m} \left(T_i - \sum_{k=1}^{p} g_{ik} \times x_{ik} \right) & (5)\\
\\
& \sum_{k=1}^{p} x_{ik} \le 1, \forall i = 1..m & (6)\\
& \sum_{i=1}^{m} x_{ik} \le m \times x_k, \forall k = 1..p & (7)\\
& x_k \le \sum_{i=1}^{m} x_{ik}, \forall k = 1..p & (8)\\
\\
& x_k \in \{0,1\}, \forall k = 1..p \\
& x_{ik} \in \{0,1\}, \forall i = 1..m, \forall k = 1..p
\end{cases}
$$

Constraints *(6)* set that query Q_i uses at most one view. Constraints *(7)* set that view V_k must be materialized if at least one query uses view V_k (i.e., it exists at least one query Q_i such that $x_{ik} = 1$). Constraints *(8)* set that view V_k must not be materialized if not used by any query.

This problem is NP-hard, because the Knapsack Problem can be identified as a subproblem: the views can be seen as the items to pack in the knapsack, each view having a weight (the cost induced by its materialization) and a profit (the gain on processing time induced by its materialization). The problem can be solved by exact methods, notably with Branch-and-Bound. However, for large instances, such methods could not provide an optimal solution in a limited time. The remainder of the article focuses on developing and analyzing a GRASP heuristic for the studied problem.

3. GRASP Metaheuristic

GRASP (*Greedy Randomized Adaptive Search Procedure*) is a metaheuristic with two phases: an iterative construction and a local search, Feo (1995), Resende (2003). This method is multi-start, as the two phases are repeated several times, and the best solution of all iterations is kept. In the construction phase, which is a greedy approach, a solution is iteratively constructed, by adding one element at a time in the solution. Then, the local search iteratively improves the solution obtained in the first phase by moving from solution to solution in the space of candidate solutions.

At each iteration of the construction phase, the candidate elements to be inserted are ranked, based on a *greedy function* that estimates the benefit of inserting an element into the solution. In order to build a different solution at each iteration of GRASP, the best candidate elements are placed in a *restrictive candidate list* (RCL), and one of them is randomly selected to be inserted into the solution. The heuristic is said to be adaptive because the ranking of the candidate elements is updated at each iteration of the construction phase to reflect the changes from the selection of the previous element.

3.1. General Description

There are two sets of decision variables in optimization problem *(P)*: x_k that indicates whether view V_k is materialized, and x_{ik} that indicates whether view V_k is used by query Q_i. Note that if all x_k are fixed, then finding the optimal values for all x_{ik} is straightforward, because of the following statements:

- Each query can use only one view;
- Queries can only use the views that are materialized;
- A view can be used by several queries;
- The objective is to minimize T_{proc}.

```
1.  function GRASP(instance):
2.      x* ← 0;
3.      T_proc* ← +∞;
4.
5.      for i = 1..IT_GRASP do
6.          greedyConstruction(instance, x);
7.          localSearch(instance, x);
8.          compute T_proc of solution x;
9.
10.         if T_proc < T_proc* then
11.             x* ← x;
12.             T_proc* ← T_proc;
13.         end if;
14.     end for;
15.
16.     return x*;
17. end function;
```

Figure 1: GRASP description.

Therefore, selecting the materialized view V_k that maximizes gain g_{ik} for each query Q_i provides an optimal solution for x_{ik}, once x_k are fixed. In our GRASP heuristic, a solution is represented by vector $x = (x_k)_{k=1..p}$. Let x^* be the best solution found so far, and T_{proc}^* the associated processing time. Figure 1 shows the main structure of the GRASP heuristic.

3.2. Greedy Construction of Randomized Solutions

The construction phase of GRASP, described in Figure 2, starts with no materialized view, i.e., with solution $x = 0$. Then, iteratively, one view is selected to be materialized in the solution. The aim here is to generate a feasible solution, meaning that the cost C_{total} of final solution x must be less than C_{max}. Therefore, only views that reduce cost C_{total} are inserted in the solution, and the candidate views are ranked according to the reduction of cost implied by their materialization. For this purpose, several indicators are introduced to estimate the impact of materializing a view.

Let c_k be the cost of materializing view V_k, which includes processing the materialization and maintenance of the view, and storing it:

$$c_k = c_c \times T_k + c_s \times T \times s_k \qquad (9)$$

Let g_k be the gain on total response time T_{proc} of materializing view V_k, which is the sum of all the gains implied by view V_k on each query Q_i:

$$g_k = \sum_{i=1}^{m} \max\left\{0, g_{ik} - \max_{l=1..p,\ x_l=1}\left\{g_{il}\right\}\right\} \qquad (10)$$

Let w_k be the benefit of materializing view V_k, which is the difference between the cost of the gain on response time and the cost of materializing the view:

$$w_k = c_c \times g_k - c_k \qquad (11)$$

In the construction phase of GRASP, the candidate views are ranked according to w_k. Only the views with $w_k > 0$ are considered, and among the best of these candidates (the best *RCL_PERC %*), one is chosen randomly. This selected view is materialized in solution x, and the procedure repeats, until there is no more candidate view to add in the solution. The cost C_{total} of solution x is computed in order to check whether x is feasible. If not, a new attempt to build a feasible solution is performed. If, after a given number of attempts *IT_GREEDY*, no feasible solution is found, then GRASP stops without providing a feasible solution.

```
1.  function greedyConstruction(instance, x):
2.      i ← 0;
3.
4.      repeat
5.          x ← 0;
6.
7.          repeat
8.              RCL ← ∅;
9.
10.             for all k = 1..p such that x_k = 0 do
11.                 compute w_k;
12.                 if w_k > 0 then insert V_k into RCL;
13.             end for;
14.
15.             if RCL ≠ ∅ then
16.                 sort RCL by descending order of w_k;
17.                 randomly select V_k in the first RCL_PERC % of RCL;
18.                 x_k ← 1;
19.             end if;
20.         until RCL = ∅;
21.
22.         compute cost C_total of solution x;
23.         i ← i + 1;
24.     until C_total ≤ C_max or i = IT_GREEDY;
25.
26.     if C_total > C_max then stop; // No feasible solution.
27. end function;
```

Figure 2: Greedy construction.

3.3. Local Search

The first step of GRASP was to build a feasible solution, i.e., to get a solution with $C_{total} \leq C_{max}$. In the second step, the local search described in Figure 3, the aim is to improve the solution by reducing total processing time T_{proc}. The procedure moves from solution to solution by adding a view to be materialized at each iteration. For this purpose, the indicator g_k of each view V_k that is not materialized yet is computed. Neighborhood solutions of x will be solutions with one more materialized view V_k such that $g_k > 0$, and that are still feasible, i.e., such that $C_{total} - w_k \leq C_{max}$. The heuristic moves to the solution that is randomly selected among the *NGB_PERC %* best solutions (i.e., with highest g_k) of the neighborhood.

```
1.  function localSearch(instance, x):
2.      repeat
3.          remove useless views from x;
4.          compute cost C_total of solution x;
5.          NGB ← ∅;
6.
7.          for all k = 1..p such that x_k = 0 do
8.              compute g_k and w_k;
9.              if g_k > 0 and C_total − w_k ≤ C_max then insert V_k into NGB;
10.         end for;
11.
12.         if NGB ≠ ∅ then
13.             sort NGB by descending order of g_k;
14.             randomly select V_k in the first NGB_PERC % of NGB;
15.             x_k ← 1;
16.         end if;
17.     until NB = ∅;
18. end function;
```

Figure 3: Local search.

Note that each time a new view is materialized, it can make some already materialized views useless, meaning that there can exist materialized views that are not used anymore by any query. To not materialize such views reduces total cost without increasing total processing time. Therefore, such views are detected and removed from each new solution of the local search. The procedure ends when no more view can be added to improve the solution.

4. Computational Results

The performance and the quality of the solutions of our GRASP heuristic are compared with those obtained by solving the optimization problem *(P)* with CPLEX 12.4. The experiments were performed on a quad-core AMD Opteron 2.3 GHz processor, with 256 GB of RAM. We choose to use the pricing of Amazon EC2 and S3 services, where the price c_s for each GB of stored data per month is 0.12, the price c_t for each GB of downloaded data is 0.1, and the price c_c for each hour of CPU processing is 0.08. We consider an operating period of one month in the cloud, i.e., $T = 1$.

We choose to test randomly generated instances, because for now, it is not possible to estimate the gains g_{ik} without running experiments on an actual cloud to measure the values, Perriot (2003). Thus, values S_D, S_A, s_k, T_i, T_k and g_{ik} are randomly generated for each instance. For this study, instances of various sizes (i.e., with different number m of queries, and number k of candidate views) are built. 24 different sizes are tested, with 5 instances each time, which makes 120 generated instances.

Moreover, we test the same instances for three different values of C_{max}: C_1 is 3 % above the minimum cost C^- of the instance, C^- is obtained by solving problem *(P)* without the budget limit and with the objective of minimizing C_{total}; C_2 is 50 % above C^-; C_3 is 50 % below the maximum cost C^+ of the instance, C^+ is obtained by solving problem *(P)* without the budget limit. Therefore, three groups of 120 instances are formed, denoted G_1, G_2, and G_3, with C_{max} respectively equal to C_1, C_2, and C_3.

For each group, two tables are presented: on the left, the number of views is fixed to 500, and the number of queries varies; and on the right, the number of queries is fixed to 500, and the number of views varies. The presented results are average values from 5 different instances with the same size. They show the time needed by GRASP and CPLEX to solve the problem (CPU time) and the relative difference (gap) between the values of the solutions found by both methods. Note that if CPLEX did not find the optimal solution within the hour, the gap is computed with the lower bound found by CPLEX.

The GRASP heuristic performs 100 iterations (*IT_GRASP = 100*) for small instances (i.e., up to 100 views and queries), and 50 iterations for large instances. The other parameters of GRASP have the following values: *IT_GREEDY = 2000*, *RCL_PERC = 10 %*, and *NGB_PERC = 10 %*.

Table 1: Results of G_1, with p = 500.

Queries	CPU Time (s)		Gap
(*m*)	CPLEX	GRASP	(%)
10	1.06	0.36	0.40
20	7.24	0.93	0.00
30	26.85	1.53	0.01
40	60.84	2.58	0.13
50	929.95	3.27	0.22
60	615.23	4.44	0.62
70	2 288.75	5.12	0.15
80	970.43	6.23	0.00
90	3 018.81	6.81	0.49
100	2 681.01	7.14	0.00
200	3 600.00	20.77	* 4.81
500	3 600.00	197.60	* 2.26

** No optimal solution found.*

Table 2: Results of G_1, with m = 500.

Views	CPU Time (s)		Gap
(*p*)	CPLEX	GRASP	(%)
10	5.14	0.15	0.00
20	63.55	0.33	0.00
30	47.21	0.71	0.00
40	179.78	1.27	0.00
50	341.71	1.38	0.04
60	676.54	2.37	0.09
70	941.05	3.44	0.13
80	2 486.50	4.65	0.02
90	1 846.51	5.08	0.02
100	2 716.25	3.74	0.02
200	3 600.00	7.60	* 2.49
500	3 600.00	197.60	* 2.26

** No optimal solution found.*

The results for group G_1 are presented in Tables 1 and 2. For many instances, the GRASP heuristic finds a solution very close to the optimal solution (less than 1 % gap). For the largest instances (above 100 views or queries), the MIP solver was not able to find the optimal solution within one hour, thus the gap is computed between the lower bound of CPLEX and the value of the solution of GRASP.

Table 3: Results of G_2, with p = 500.

Queries	CPU Time (s)		Gap
(*m*)	CPLEX	GRASP	(%)
10	0.75	0.60	0.00
20	39.76	0.13	0.00
30	122.72	0.25	0.00
40	505.34	0.22	0.00
50	311.65	0.33	0.00
60	116.07	0.57	0.00
70	93.89	0.56	0.00
80	467.63	0.68	0.00
90	348.80	0.76	0.00
100	778.89	0.81	0.00
200	2 841.50	4.68	0.00
500	1 172.07	3.60	0.00

Table 4: Results of G_2, with m = 500.

Views	CPU Time (s)		Gap
(*p*)	CPLEX	GRASP	(%)
10	0.15	0.05	0.00
20	0.45	0.10	0.00
30	8.10	0.69	0.03
40	11.85	1.02	0.21
50	56.91	1.52	0.03
60	401.09	2.09	0.03
70	834.86	2.53	0.03
80	2 106.82	4.25	0.07
90	2 399.24	5.50	0.02
100	2 921.51	1.39	0.01
200	1 499.65	1.31	0.01
500	1 172.07	3.60	0.00

The results for group G_2 are presented in Tables 3 and 4. GRASP is efficient on this kind of instances, as its running time never exceeds 6 seconds and is always very close, if not equal, to the optimal solution. With the budget limit loosened, it seems that good solutions are easier to find for the GRASP heuristic.

Table 5: Results of G_3, with p = 500.

Queries (*m*)	CPU Time (s) CPLEX	CPU Time (s) GRASP	Gap (%)
10	0.31	0.05	0.00
20	0.74	0.12	0.00
30	0.84	0.17	0.00
40	1.26	0.23	0.00
50	1.33	0.27	0.00
60	1.49	0.54	0.00
70	1.60	0.60	0.00
80	1.70	0.68	0.00
90	1.81	0.77	0.00
100	2.21	0.85	0.00
200	4.65	1.61	0.00
500	239.93	3.59	0.00

Table 6: Results of G_3, with m = 500.

Views (*p*)	CPU Time (s) CPLEX	CPU Time (s) GRASP	Gap (%)
10	*	*	*
20	*	*	*
30	*	*	*
40	177.27	7.10	0.04
50	173.42	1.56	0.09
60	368.33	2.06	0.03
70	276.14	1.73	0.03
80	168.89	0.94	0.00
90	22.55	0.67	0.00
100	16.96	1.09	0.00
200	54.58	1.36	0.00
500	239.93	3.59	0.00

** No feasible solution.*

The results for group G_3 are presented in Tables 5 and 6. GRASP is also efficient on this kind of instances. Note that in Table 5, the instances seem to be not very hard to solve, meaning that the budget limit may not be very constraining. At the opposite, for small instances of Table 6, the budget limit (set to 50 % of C^+) is too restrictive to find any feasible solution.

5. Conclusion

This paper proposes a GRASP heuristic to optimize the materialization of views for a database stored in the cloud. This problem can be formulated as a mixed integer program, but for large instances, MIP solvers can not always provide an optimal solution in a limited time. Experiments on randomly generated instances have been achieved, showing that the GRASP heuristic can provide good solutions (usually below 1 % gap from the optimal solution) in a short running time (a few seconds to less than 4 minutes depending on the instance).

Experiments on concrete instances are now necessary to confirm that this approach is of interest for cloud computing. However, many issues remain to be tackled for the method to be usable on large scale databases: (i) for now, processing times are estimated on actual clouds, therefore formulations of those times (with and without the use of materialized views) are necessary to avoid long experiments; (ii) the problem studied here has to be completed, because for now, candidate views are given, whereas they should be identified through the optimization process.

Acknowledgements

This work has been achieved through the cooperation BRAFITEC between the Department of Production Engineering of the Federal University of Minas Gerais (UFMG) and the French school ISIMA (Institut Supérieur d'Informatique, de Modélisation et de leurs Applications) located in the city of Clermont-Ferrand.

References

Baril, X, and Bellahsène, Z. (2003), Selection of Materialized Views: a Cost-Based Approach, *CAiSE 2003, LNCS 2681*, 665-680.

Feo, T. A., and Resende, M. G. C. (1995), Greedy Randomized Adaptive Search Procedures, *Journal of Global Optimization*, 6, 109–133.

Nguyen, T., d'Orazio, L., Bimonte, S., and Darmont J. (2012), Cost Models for View Materialization in the Cloud, *Proceedings of the 2012 Joint EDBT/ICDT Workshops*, 47-54.

Perriot, R., Pfeifer, J., d'Orazio, L., Bachelet B., Bimonte, S., and Darmont J. (2013), Modèles de coût pour la sélection de vues matérialisées dans le nuage, application aux services

Amazon EC2 et S3, *9th French-Speaking Workshop on Data Warehousing and Online Analysis (EDA'13)*, to appear.

Resende, M. G. C., and Ribeiro, C. C. (2003), Greedy Randomized Adaptive Search Procedures, *Handbook of Metaheuristics*, Kluwer Academic Publishers, 219-249.

Yang, J., Karlapalem K., and Li Q. (1997), Algorithms for Materialized View Design in Data Warehousing Environment, *Proceedings of the 23rd VLDB Conference*, 136-145.