Modeling view selection as a constraint satisfaction problem

Imene Mami, Remi Coletta, and Zohra Bellahsene

LIRMM, University Montpellier 2
161 Rue Ada
F-34095 Montpellier, France
{imen.mami,coletta,bella}@lirmm.fr

Abstract. Using materialized views can highly speed up the query processing time. This paper deals with the view selection issue, which consists in finding a set of views to materialize that minimizes the expected cost of evaluating the query workload, given a limited amount of resource such as total view maintenance cost and/or storage space. However, the solution space is huge since it entails a large number of possible combinations of views. For this matter, we have designed a solution involving constraint programming, which has proven to be a powerful approach for modeling and solving combinatorial problems. The efficiency of our method is evaluated using workloads consisting of queries over the schema of the TPC-H benchmark. We show experimentally that our approach provides an improvement in the solution quality (i.e., the quality of the obtained set of materialized views) in term of cost saving compared to genetic algorithm in limited time. Furthermore, our approach scales well with the query workload size.

1 Introduction

The information stored at the warehouse is often organized in materialized views which represent pre-computed portions of the most frequently asked queries [3]. Using materialized views can improve the performance and speed up the processing of queries since the access to materialized views can be much faster than recomputing the views. However, these materialized views have to be maintained in response to changes to the underlying base relations. In most cases it is wasteful to maintain a view by re-computing it from scratch. Often, it is cheaper to compute only the changes in the view to update its materialization which is called the incremental view maintenance.

Materializing all the input queries can achieve the lowest query cost but the highest view maintenance cost which can cause overhead to the system. Besides, the query result can be too large to fit in the available storage space. Hence, there is a need for selecting a set of views to materialize by taking into account three important features: query cost, view maintenance cost and storage space.

The problem of choosing which views to materialize that minimize the total query cost given a limited amount of resource such as total view maintenance cost and/or storage space is known as the view selection problem. This is one of the most challenging problems in data warehousing [28]. The view selection problem is a NP-complete problem since the search space for the optimal solution grows exponentially as the problem size increases [11,12].

In this paper, we propose a new approach to the view selection problem which minimizes the total query cost under the case where (i) the limited resource is the total view maintenance cost, assuming unlimited amount of storage space if we consider that storage space is cheap and not regarded as a critical resource anymore and (ii) both space and maintenance cost constraints exist.

Although, several heuristic algorithms have been proposed in literature to solve the view selection problem such as deterministic algorithms i.e., greedy algorithms [11,29,12,25,20,21,26,2], randomized algorithms i.e., genetic algorithms [31,18,13,30,16,5] and simulated annealing algorithms [14,7,8] or hybrid algorithms [32] which combine the strategies of pure deterministic algorithms and pure randomized algorithms.

These heuristic algorithms provide reasonably good solutions. However, there is no guarantee of performance because the greedy nature or the random characteristic of the algorithms may make them converging to poor local minima. An exact resolution for the view selection problem is prohibited since an exhaustive search cannot compute an optimal solution within a reasonable time due to the complexity of the problem.

Yet, over the past ten years, effective paradigms for exact resolution of NP-complete problems have been proposed, such as constraint programming (CP), structured around annual competitions [17]. Furthermore, constraint programming has proven to be a powerful technique for modeling and solving combinatorial problems [9]. We have designed a new approach for solving the view selection problem involving constraint programming. Our approach consists in modeling the view selection as a Constraint Satisfaction Problem (CSP). Our main contributions are:

1. We propose a new approach to the view selection problem. We model this problem as a constraint satisfaction problem (CSP). Then, a constraint programming (CP) solver can be applied to set up the search space by identifying a set of views that minimizes the total query cost.
2. We address the view selection under the case where (i) the limited resource is the total view maintenance cost, assuming unlimited amount of storage space if we consider that storage space is cheap and not regarded as a critical resource anymore and (ii) both space and maintenance cost constraints exist.
3. We highlight the *anytime* behavior of our approach which is able to provide a near optimal solution to the view selection problem during a given time interval. The quality of this solution may be improved over time (if the CPU time is available).
4. We have implemented our approach and compared it with a randomized method (i.e., genetic algorithm). We experimentally show that our approach provides better performance resulting from evaluating the quality of the solutions in term of cost savings and scales well with the query workload.

The rest of this paper is organized as follows. Section 2 describes the problem of view selection. Section 3 provides an overview of our approach and describes how to model the view selection problem as a constraint satisfaction problem (CSP). In section 4, are provided the experiments results. Section 5 contains a brief survey of related work. Finally, in section 6 we conclude and plan for future work.

2 Problem specification

The general problem of view selection is to select a set of views to be materialized that minimizes the cost of evaluating the query workload modeled by the frequently asked queries, given a limited amount of resource, e.g., total view maintenance cost and/or storage space. In this paper, we consider selection-projection-join (SPJ) queries that may involve aggregation and group by clause as well.

In order to detect overlapping between queries of workload and capture the dependencies among the queries, the view selection is represented as an AND-OR view graph [11]. The union of all possible execution plans of each query forms an AND-OR view graph where the common sub-expressions are represented once. The AND-OR view graph is a Directed Acyclic Graph (DAG) composed of two types of nodes: Operation nodes (Op-nodes) and Equivalence nodes (Eq-nodes). Each Op-node represents an algebraic expression (Select-Project-Join) with possible aggregate function. An OR-node represents a set of logical expressions that are equivalent (i.e., that yield the same result). The Op-nodes have only Eq-nodes as children and Eq-nodes have only Op-nodes as children. The root nodes are equivalence nodes representing the queries of workload and the leaf nodes represent the base relations. Equivalence nodes in the AND-OR view graph correspond to the views that are candidates for materialization.

A sample AND-OR view graph is shown in figure 1. Circles represent operation nodes and boxes represent equivalence nodes. For example, in this figure, view v_1, corresponding to a single query, can be computed from v_3 and r_2 or v_4 and r_3. If there is only one way to answer or update a given query, the graph becomes an AND view graph. We explore the view selection problem in the context of AND-OR view graph, which allows a single query to be answered and updated from multiple paths, since a good selection of materialized views can only be found by considering the optimization of both global processing plans and materialized view selection [32].

To each equivalence node which represents a view, is associated the following metadata:

- Query cost Qc which represents the cost of computing a view from its related base relations and/or views.
- Maintenance cost Mc which is the cost required for updating a view when the related base relation is changed.
- Read cost Rc that denotes the size of the view.

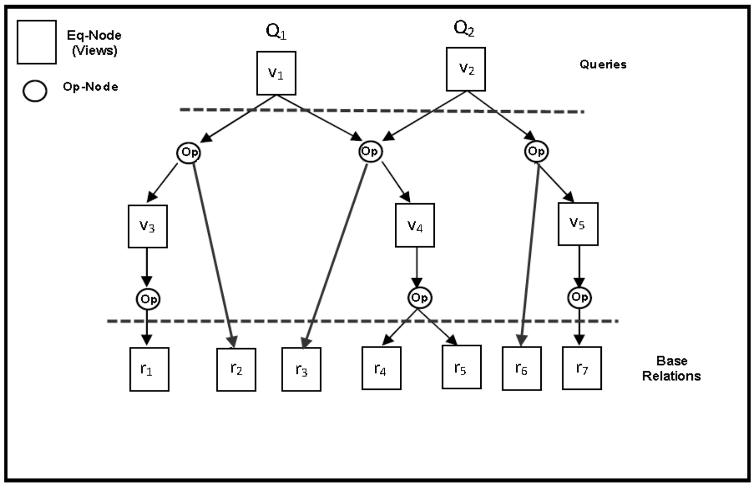


Fig. 1: The AND-OR view graph of the two queries Q_1 and Q_2.

- Query frequency f_q (if the equivalence node is a root node) which describes the frequency of posing a given query.
- Update frequency f_u which represents the frequency of updating a view in response to change to the underlying data.

The general view selection problem for AND-OR view graphs can be formulated as follows: Given an AND-OR view graph G, select the set of views to materialize that minimizes the total query cost under the following cases:

- Where only the maintenance cost constraint is considered. Here, we address the view selection problem by constraining the total maintenance cost needed to update the materialized views while assuming unlimited amount of storage space.
- Where both maintenance cost and space constraints exist. In this case, we have a total maintenance cost limit but also a bound on the total storage space required to materialize the selected views.

3 A new approach to the view selection problem

3.1 Motivations

In this section, we present our approach for selecting a set of views to be materialized. The optimal solution is the one which selects the right materialized views (equivalence nodes) of the AND-OR view graph that minimizes the total query cost subject to certain constraints such as space and maintenance cost constraints. However, the search space for the optimal solution is very large since it entails a great number of comparisons between all possible subsets of views.

Our motivation to use constraint programming in solving the view selection problem is that it is known to be a powerful approach for modeling and solving combinatorial problems such as Job Shop Scheduling [4]. The success of using constraint programming for combinatorial optimization is due to its combination of these three features [27]:

– *High level modeling.* Constraint programming provides a rich constraint language to model the problem as a Constraint Satisfaction Problem. In the following subsections, we give a formal definition of CSP and show how to model the view selection problem as a CSP.
– *Constraint propagation.* This leads to a reduction of the search space, by excluding solutions where the constraints become inconsistent. For example, in the case of solving the view selection problem under the maintenance cost constraint, all the view combinations which violate this constraint are discarded.
– *Search.* Constraint programming offers facilities to control the search behavior deciding which alternative (i.e., views combination) to try first.

3.2 Preliminaries

In this subsection, we introduce the CSP model that we have used in our approach. A constraint satisfaction problem (CSP) is composed of:

– A set of variables $VAR = \{var_1, var_2, ..., var_n\}$
– Each variable var_i has a set of values which is called the domain of values $DOM = \{d_1, d_2, ..., d_n\}$
– A set of constraints $CST = \{c_1, c_2, ..., c_n\}$ describes the relationship between subsets of variables. Formally, a constraint C_{ijk} between the variables var_i, var_j, var_k is any subset of the possible combinations of values of var_i, var_j, var_k, i.e., $C_{ijk} \subset d_i \times d_j \times d_k$. The subset specifies the combinations of values that the constraint allows.

A CSP consists in finding solutions by assigning values to its variables that satisfy all its constraints. Our approach consists in modeling the view selection problem as a CSP. Its resolution is supported automatically by constraint solver embedded in the constraint programming language.

3.3 Modeling View Selection Problem as a CSP

Formulating the view selection problem as a constraint satisfaction problem (CSP) consists in specifying the variables of the CSP, their domains, and the constraints that are over them in the context of view selection. In the following, we describe each part of the specification.

3.3.1 Variables & Domains

The variables of the CSP considered in modeling the view selection problem are:

– Mat_{v_i} which denotes for each view v_i (equivalence node in the AND-OR view graph) , if it is materialized or not materialized. It is a binary variable, $d_{Mat_{v_i}}$ =0,1 (0: v_i is not materialized, 1: v_i is materialized).
– $Qc(v_i)$ that represents the query cost corresponding to a view v_i. The domain is a finite subset of $\mathbb{N}^*$ such as $d_{Qc(vi)} \subset \mathbb{N}^*$.
– $Mc(v_i)$ which is the maintenance cost corresponding to a view v_i, where $d_{Qc(vi)} \subset \mathbb{N}^*$.

3.3.2 Constraints

The constraints which describe the relationship between the variables defined above are:

– The query and maintenance cost corresponding to a view are implemented by using a depth-first traversal of the AND-OR view graph. We use the cost formulae described in [24,21] to compute these two costs (defined in a recursive way).

Query Cost:

$$Qc(v_i) = \begin{cases} ComputingCost(v_i) \;\; if \;\; Mat_{v_i} = 0 \\ \min(ComputingCost(v_i), ReadingCost(v_i)) \; otherwise \end{cases}$$

$$ComputingCost(v_i) = \min_{op_j \in child(v_i)} \left(cost(op_j) + \sum_{v_k \in child(op_j)} Qc(v_k) \right)$$

The query cost corresponding to view v_i which is an equivalence node in the AND-OR view graph, is the minimum cost paths from v_i to its related views or base relations, if v_i is not materialized. Otherwise, if v_i is materialized, we use the minimum of the cost of reading v_i and the minimum cost paths as defined above.
Each minimum cost path is composed of all the cost of executing the operation nodes on the path and the query cost corresponding to the related views or bases relations of v_i. The costs of executing the operations: selection, join, projection and aggregation are estimated according to the formulas given in [6] for cost operation estimation. In this paper, these costs are calculated in terms of number of tuples in the involved relations.

View maintenance cost:

$$Mc(v_i) = \begin{cases} 0 \;\; if \;\; Mat_{v_i} = 0 \\ \sum_{dr_l \in diffRelations(v_i)} Mcost(v_i, dr_l) \; otherwise \end{cases}$$

$$Mcost(v_i, dr_l) = \min_{op_j \in child(v_i)} \left(cost(op_j, dr_l) + \sum_{v_k \in child(op_j)} UpdatingCost(v_k, dr_l) \right)$$

$$UpdatingCost(v_k, dr_l) = \begin{cases} Mcost(v_k, dr_l) \;\; if \;\; Mat_{v_k} = 0 \\ \min(Mcost(v_k, dr_l), \delta(v_k, dr_l)) \; otherwise \end{cases}$$

The maintenance cost of view v_i, if it is materialized, is computed by summing the number of changes in the base relations from which v_i is updated. If v_i is not materialized, then there is no maintenance cost. We assume incremental maintenance to estimate the view maintenance cost. Therefore, the maintenance cost is the differential results of materialized views given the differential (updates) of the bases relations. Let $\delta(v_i, dr_l)$ denotes the differential result of view v_i, with respect to update dr_l.
The view maintenance cost is computed similarly to the query cost, but the cost of each minimum path is composed of all the cost of executing the operation nodes with respect to update dr_l on the path and the maintenance cost corresponding to the related views of v_i. We have been inspired by the formula given in [21] for estimating the cost of executing the operation nodes in response to changes to the base relations.

– The total maintenance cost of the set of materialized views is less than U which is the total view maintenance cost limit.

$$\sum_{v_i \in V(G)} (Mat_{v_i} * (f_u(v_i) * Mc(v_i))) \leq U$$

Note that $V(G)$ represents all the views in the AND-OR view graph, $Mc(v_i)$ is the cost of maintaining a materialized view v_i and $f_u(v_i)$ is the update frequency of the view v_i. Here, the view selection is decided under the maintenance cost constraint.

– The total space occupied by the materialized views M is less than S which is the maximum storage space.

$$\sum_{v_i \in V(G)} (Mat_{v_i} * Rc(v_i)) \leq S$$

Recall that $Rc(v_i)$ is the size of the view v_i. If the space constraint is considered, views are selected to be stored only if the necessary space for their materialization is at most S.

– Minimize the total query cost.

$$minimize \left(\sum_{v_i \in Q(G)} (f_q(v_i) * Qc(v_i)) \right)$$

In this formula, $Q(G)$ represents all the queries (root nodes in the AND-OR view graph), $Qc(v_i)$ is the query cost of the view v_i and $f_q(v_i)$ is the query frequency of the view v_i. The total query cost is computed by summing over the cost of processing all the queries of workload.

4 Experimental Evaluation

We have implemented our approach and compared it with a randomized method because in previous studies [30,7,8], it was shown that randomized algorithms provide a significant improvement in the performance compared to deterministic algorithms. The most commonly used randomized algorithms in the context of view selection are simulated annealing algorithms and genetic algorithms. The motivation to compare our approach with genetic algorithm is based on the observation that genetic algorithm in contrast with simulated annealing algorithm use a multi-directional search which allows to find a point near the global optimum [18]. A comparison with hybrid approaches has not been made because of their excessive computation time [32]. The goal of the experiments is the comparison of the solution quality resulting from evaluating the quality of the obtained set of materialized views in term of cost savings between our method and genetic algorithm.

4.1 Experiment Settings

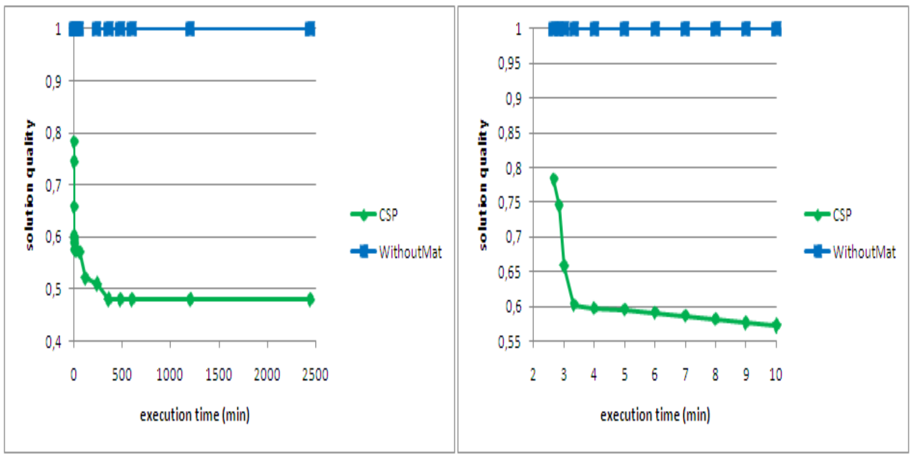
The computer used for experimentation was an Intel Core 2 Duo P8600 CPU @ 2.40 GHz machine running with 3GB of RAM and Windows XP Professional SP3. The program was written in Java using JDK/JRE 1.6.0. We chose a workload of one hundred queries defined over the database schema of the TPC-H benchmark [1]. We then randomly assigned values to the frequencies for access and update based on uniform distribution.

In all experiments, the quality of the solutions found by the genetic algorithm and our method was respectively measured as a ratio of the total query cost obtained using the genetic algorithm and the constraint solver over the total query cost when all the views are not materialized. Thus, we consider the "Without-Mat" approach which does not materialize views and always recomputes queries as a benchmark for our normalized results. The ratio was computed and averaged over several runs for the genetic algorithm because of his probabilistic behavior.

We have implemented the genetic algorithm presented in [5] by incorporating space and maintenance cost constraints into the algorithm. The values for population size and probabilities of the crossover and mutation operators are assigned based on studies conducted by [10,19]. In order to let the genetic algorithm converge quickly, we generated an initial population which represents a favorable view configuration rather than a random sampling. Favorable view configuration such as the views which satisfy the maintenance cost and space constraints if they exist are most likely selected for materialization.

4.2 Experimental Results

The constraint solver for solving the view selection problem as a constraint satisfaction problem (CSP) is aimed to provide the optimal solution (see figure 2(a)). However, this is not feasible at large scale because of the great number of comparisons between all possible subsets of views which are candidate for materialization. One way to avoid an explosion in the search space is to explore a strictly limited number of possibilities. In this case, the constraint solver is aimed at simply finding a feasible solution in which all constraints are satisfied. Figure 2(b) shows the quality of the solutions returned by our approach involving 30 queries as a function of execution time. We can see, our approach provides near optimal solutions quickly. Indeed, after only few minutes (i.e., 3 minutes and 24 seconds), the solution found lies within 79,84% of the optimal solution. The quality of this solution is improved over time. In the next experiments, the constraint solver was left to run until the convergence of the genetic algorithm.



(a) Until optimal solution (b) Until 10 minutes

Fig. 2: The solution quality of our approach over time (nb of queries=30).

4.2.1 Experiments under the maintenance cost constraint In this section, we compare the performance resulting from evaluating the quality of the solutions found by our approach and genetic algorithm in the context where the view selection is constrained by a total view maintenance cost limit U. In these experiments, U is computed as a function of $U(Q)$ which is the total maintenance cost when all queries (i.e., root nodes in the AND-OR view graph) are materialized [14]. We set U to be 4%, 8%, 16%,..., 32% of $U(Q)$. Figure 3 compares experiment results of our approach with that of genetic algorithm while

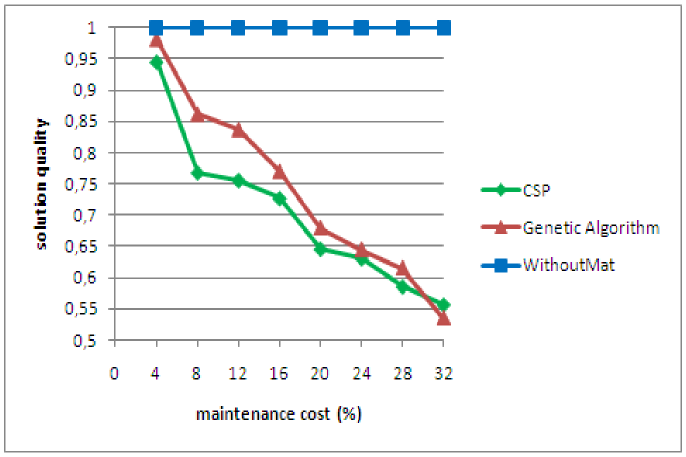


Fig. 3: Comparison of the view selection methods while varying the maintenance cost constraint (nb of queries=70).

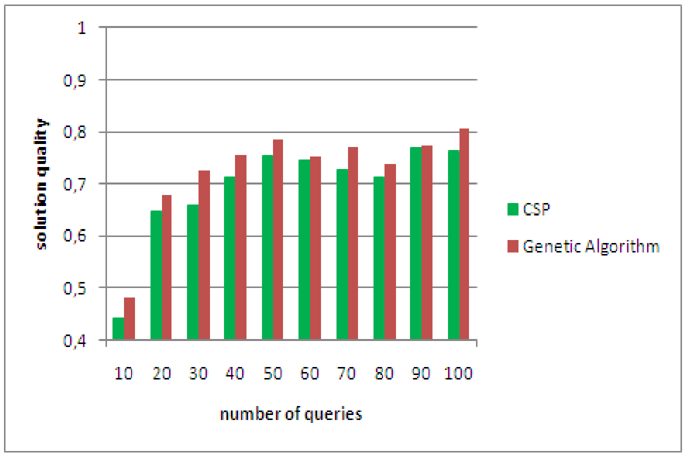


Fig. 4: Evaluating the performance while varying the number of queries (U=16% of $U(Q)$).

varying the values of U for a workload involving 70 queries. We can see in figure 3 that our approach generates solutions with cost less than that returned by the genetic algorithm when $U < 32\%$ of $U(Q)$. For large values of U ($U >= 32\%$ of $U(Q)$), solutions returned by the genetic algorithm have better quality compared with our approach. This is because the genetic algorithm converges faster when we relax the maintenance cost constraint and our approach needs more time to achieve better quality. However, this would not be seen as a drawback since the time bound for the update is usually very tight compared with the time required for maintaining all the query workload. Figure 4 shows the costs of the workload involving 10,20,30,40,50,60,70,80,90 and 100 queries in order to test the scalability of the view selection methods according to the number of queries. The maintenance cost constraint U was set to 16% of $U(Q)$. This graphics shows that our approach provides an improvement in the quality of the obtained set of materialized views in term of cost savings compared with the genetic algorithm. Furthermore, our method supports scalability when the number of queries increases.

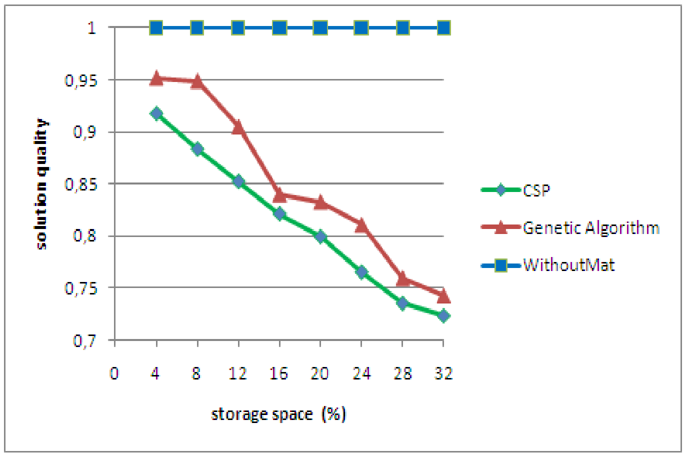


Fig. 5: The solution quality as a function of available storage space (number of queries=70, U=50% of $U(Q)$).

4.2.2 Experiments under the maintenance cost and space constraints

In order to compare the performance of the view selection methods in the context where both maintenance cost and space constraints exist, we set U to 50% and give restrictive values to the space constraint S. In these experiments, S is computed as a function of $S(Q)$ which is the size of the whole workload [14]. We varied S between 4% and 32% of $S(Q)$. Figure 5 illustrates the quality of the solutions produced by the two methods for various values of S for a workload involving 70 queries. The graphic shows that our approach generates better

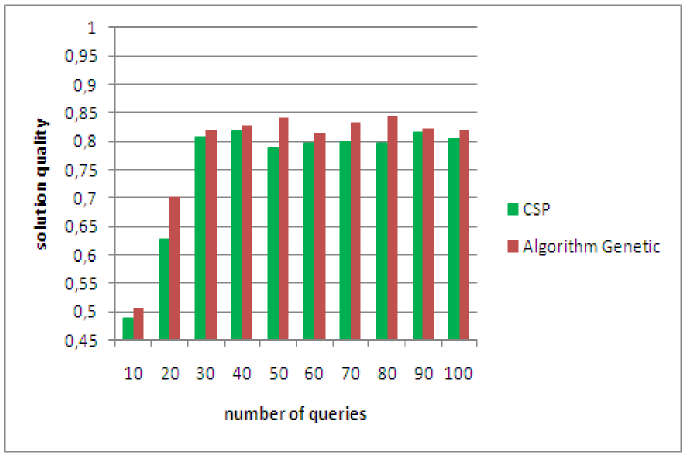


Fig. 6: The quality of results as a function of the number of queries (U=50% of $U(Q)$, S=20% of $S(Q)$).

solutions than the genetic algorithm in the case where the storage space is the restrictive constraint. Experiment results depicted in figure 6 shows how well the view selection methods scale with the problem size. The maintenance cost constraint was set to 50% of $U(Q)$ and the space constraint to 20% of $S(Q)$. The size of the problem varied from 10 to 100 queries. We observe that our approach provides the lowest query cost while varying the number of queries. Therefore, our approach outperforms the genetic algorithm for all query workload size. We also show that our approach can be applied to large number of queries.

5 Related Work

As mentioned in the introduction, the view selection problem is a NP-complete problem. Several view selection methods have been proposed in the literature to address the view selection problem. They can be classified into three major groups:

5.1 Deterministic algorithms based methods

Such methods usually provide a solution to the view selection problem either by applying exhaustive search or by applying heuristics i.e., greedy algorithms to reduce the search space. In [23], an exhaustive approach is presented for finding the best set of views to materialize. [15] presents an optimal algorithm based on A* algorithm [22] that vastly prune the search space compared to the algorithm proposed in [23]. However, an exhaustive search cannot compute an optimal solution within a reasonable time due to the complexity of the problem. Many approaches [11,29,12,25,20,21,26,2] using a kind of greedy strategy to avoid having to traverse the solution space in an exhaustive search manner have been designed. However, greedy algorithms are unsatisfactory in term of the solution quality i.e., the quality of the obtained set of materialized views because the greedy nature of the algorithm makes it susceptible to poor local minima (initial solutions influence the solution greatly). In contrast with these work, our approach provides a suitable trade-off between the computation time and the solution quality. Indeed, we have shown that our approach provides a good solution quality in a limited time. This is due to the use of constraint propagation technique and facilities for controlling search behavior that are the features of constraint programming.

5.2 Randomized algorithms based methods

Randomize algorithms such as simulated annealing algorithms [7,8,14] and genetic algorithms [5,13,18,30,16] have been used and explored for the selection of materialized views in order to improve the quality of the solution. These algorithms use a randomized search strategy for deciding which views to materialize. Randomized algorithms provide a better solution quality than greedy algorithms. However, they may have a tendency to converge towards local optima. Besides, their successes often depend on the set-up of the algorithm as well as the extremely difficult fine-tuning of algorithm that must be performed during many test runs. In our approach, we simply model view selection as a constraint satisfaction problem (CSP) and its resolution is supported automatically by constraint solver embedded in the constraint programming language. Besides, our approach provides better results compared with genetic algorithm in term of the solution quality.

5.3 Hybrid algorithms based methods

Hybrid algorithms combine the advantages of pure deterministic algorithms and pure randomized algorithms. A hybrid approach has been applied in [32] for the view selection problem which combine heuristic algorithms i.e., greedy algorithms and genetic algorithms. They prove that hybrid algorithms provide better performance than either the genetic algorithms or heuristic algorithms used alone in terms of solution quality. However, they often require longer computation time and may be impractical due to their excessive computation time.

6 Conclusion

In this paper, we have presented a new approach to the view selection problem which is based on constraint programming. More specifically, the view selection problem has been modeled as a constraint satisfaction problem (CSP). Its resolution has been supported automatically by constraint solver embedded in the constraint programming language. We have performed several experiments and comparison with a randomized method i.e., genetic algorithm. The experiment results have shown that our approach provides better performances compared with the genetic algorithm in term of the solution quality (i.e., the quality of the obtained set of materialized views) in a limited time. More precisely, we have demonstrate experimentally that our approach provides better results compared with genetic algorithm in term of cost savings when the view selection is decided under the case where (i) only the maintenance cost constraint is considered, assuming unlimited amount of storage space and (ii) both maintenance cost and space constraints exists. We have also shown that our approach supports scalability when the number of queries increases. As a future work, we are planning to apply our approach to a distributed database setting. Our current approach simply takes into account the space and maintenance cost constraints. These constraints will be per machine in a distributed context. Also, resource constraints such us CPU, IO, network bandwidth and the location of materialized views will have to be taken into consideration. These new constraints will easily be handled with our approach.

References

1. TPC-R Benchmark Standard Specication 2.01. http://www.tpc.org, January 1999.
2. Xavier Baril and Zohra Bellahsene. Selection of materialized views: A cost-based approach. In *CAiSE*, pages 665–680, 2003.
3. Randall G. Bello, Karl Dias, Alan Downing, James J. Feenan Jr., James L. Finnerty, William D. Norcott, Harry Sun, Andrew Witkowski, and Mohamed Ziauddin. Materialized views in oracle. In *VLDB*, pages 659–664, 1998.
4. Yves Caseau and François Laburthe. Improved clp scheduling with task intervals. In *ICLP*, pages 369–383, 1994.
5. Leonardo Weiss F. Chaves, Erik Buchmann, Fabian Hueske, and Klemens Böhm. Towards materialized view selection for distributed databases. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '09, pages 1088–1099, New York, NY, USA, 2009. ACM.
6. Rada Chirkova, Alon Y. Halevy, and Dan Suciu. A formal perspective on the view selection problem. *VLDB J.*, 11(3):216–237, 2002.
7. Roozbeh Derakhshan, Frank K. H. A. Dehne, Othmar Korn, and Bela Stantic. Simulated annealing for materialized view selection in data warehousing environment. In *Databases and Applications*, pages 89–94, 2006.
8. Roozbeh Derakhshan, Bela Stantic, Othmar Korn, and Frank K. H. A. Dehne. Parallel simulated annealing for materialized view selection in data warehousing environments. In *ICA3PP*, pages 121–132, 2008.
9. Mehmet Dincbas, Helmut Simonis, and Pascal Van Hentenryck. Solving large combinatorial problems in logic programming. *The Journal of Logic Programming*, 8(1-2):75 – 93, 1990.
10. David E. Goldberg. *Genetic Algorithms in Search Optimization and Machine Learning.* Addison-Wesley, 1989.
11. Himanshu Gupta. Selection of views to materialize in a data warehouse. In *ICDT*, pages 98–112, 1997.
12. Himanshu Gupta and Inderpal Singh Mumick. Selection of views to materialize under a maintenance cost constraint. In *ICDT*, pages 453–470, 1999.

13. Jorng-Tzong Horng, Yu-Jan Chang, and Baw-Jhiune Liu. Applying evolutionary algorithms to materialized view selection in a data warehouse. *Soft Comput.*, 7(8):574–581, 2003.
14. Panos Kalnis, Nikos Mamoulis, and Dimitris Papadias. View selection using randomized search. *Data Knowl. Eng.*, 42(1):89–111, 2002.
15. Wilburt Labio, Dallan Quass, and Brad Adelberg. Physical database design for data warehouses. In *Proceedings of the Thirteenth International Conference on Data Engineering*, ICDE '97, pages 277–288, Washington, DC, USA, 1997. IEEE Computer Society.
16. Michael Lawrence. Multiobjective genetic algorithms for materialized view selection in olap data warehouses. In *GECCO*, pages 699–706, 2006.
17. Christophe Lecoutre, Olivier Roussel, and Marc R. C. van Dongen. Promoting robust black-box solvers through competitions. *Constraints*, 15(3):317–326, 2010.
18. Minsoo Lee and Joachim Hammer. Speeding up materialized view selection in data warehouses using a randomized algorithm. *Int. J. Cooperative Inf. Syst.*, 10(3):327–353, 2001.
19. Zbigniew Michalewicz. *Genetic algorithms + data structures = evolution programs (3rd ed.)*. Springer-Verlag, London, UK, 1996.
20. Hoshi Mistry, Prasan Roy, Krithi Ramamritham, and S. Sudarshan. Materialized view selection and maintenance using multi-query optimization. *CoRR*, cs.DB/0003006, 2000.
21. Hoshi Mistry, Prasan Roy, S. Sudarshan, and Krithi Ramamritham. Materialized view selection and maintenance using multi-query optimization. In *SIGMOD Conference*, pages 307–318, 2001.
22. Nils J. Nilsson. *Problem-Solving Methods in Artificial Intelligence*. McGraw-Hill Pub. Co., 1971.
23. Kenneth A. Ross, Divesh Srivastava, and S. Sudarshan. Materialized view maintenance and integrity constraint checking: Trading space for time. In *SIGMOD Conference*, pages 447–458, 1996.
24. Prasan Roy, S. Seshadri, S. Sudarshan, and Siddhesh Bhobe. Efficient and extensible algorithms for multi query optimization. *CoRR*, cs.DB/9910021, 1999.
25. Prasan Roy, S. Seshadri, S. Sudarshan, and Siddhesh Bhobe. Efficient and extensible algorithms for multi query optimization. In *SIGMOD Conference*, pages 249–260, 2000.
26. Satyanarayana R. Valluri, Soujanya Vadapalli, and Kamalakar Karlapalem. View relevance driven materialized view selection in data warehousing environment. In *Australasian Database Conference*, 2002.
27. Mark Wallace. Practical applications of constraint programming. *Constraints*, 1:139–168, 1996. 10.1007/BF00143881.
28. Jennifer Widom. Research problems in data warehousing. In *CIKM*, pages 25–30, 1995.
29. Jian Yang, Kamalakar Karlapalem, and Qing Li. Algorithms for materialized view design in data warehousing environment. In *VLDB*, pages 136–145, 1997.
30. Jeffrey Xu Yu, Xin Yao, Chi-Hon Choi, and Gang Gou. Materialized view selection as constrained evolutionary optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 33(4):458–467, 2003.
31. Chuan Zhang and Jian Yang. Genetic algorithm for materialized view selection in data warehouse environments. In *DaWaK*, pages 116–125, 1999.
32. Chuan Zhang, Xin Yao, and Jian Yang. An evolutionary approach to materialized views selection in a data warehouse environment. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 31(3):282–294, 2001.