

# A Constraint Satisfaction based Approach to View Selection in a Distributed Context

Imene Mami, Zohra Bellahsene, Remi Coletta

► **To cite this version:**

Imene Mami, Zohra Bellahsene, Remi Coletta. A Constraint Satisfaction based Approach to View Selection in a Distributed Context. BDA: Bases de Données Avancées, 2012, Clermont-Ferrand, France. 28e journées Bases de Données Avancées, 2012. <lirmm-00736723>

**HAL Id: lirmm-00736723**

**<https://hal-lirmm.ccsd.cnrs.fr/lirmm-00736723>**

Submitted on 28 Sep 2012

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# A Constraint Satisfaction based Approach to View Selection in a Distributed Context

Imene Mami<sup>1</sup>, Zohra Bellahsene<sup>1</sup>, and Remi Coletta<sup>2</sup>

<sup>1</sup>University Montpellier 2 - INRIA, LIRMM, France

<sup>2</sup>University Montpellier 2 - LIRMM, France

{mami,bella,coletta}@lirmm.fr

## Abstract

Les vues matérialisées sont utilisées dans les systèmes de gestion de bases de données commercialisés pour accélérer le temps de traitement des requêtes. Cette technique est aussi très utile dans les entrepôts de données pour améliorer les performances des requêtes décisionnelles. Cependant, le problème de sélection de vues a été étudié surtout dans un contexte centralisé. Dans cet article, nous abordons le problème de sélection de vues dans un environnement distribué. Pour ce faire, nous avons étendu le concept des graphes de vues AND-OR qui permet de capturer les caractéristiques d'un système distribué. Nous proposons ensuite une méthode utilisant la programmation par contraintes pour la modélisation et la résolution du problème de sélection de vues sous des contraintes de ressources multiples. Nous montrons expérimentalement que notre approche fournit de meilleures performances au sens où elle permet d'améliorer la qualité des solutions en réduisant davantage encore le coût de traitement des requêtes.

**Keywords:** materialized views, query processing and optimization, view selection.

## 1 Introduction

View materialization is a widely used strategy in commercial database systems and data warehousing systems to improve query performance. Indeed, answering queries using materialized views can significantly speed up the query processing since the access to materialized views is much faster than recomputing views on demand. However, whenever a base relation is changed the materialized views built on it have to be updated in order to compute up-to-date query results. The process of updating materialized views is known as view maintenance. Besides, materialized views require storage space. The problem of choosing which views to materialize by taking into account three important features: query cost, view maintenance cost and storage space is known as the view selection problem and to be a NP-complete problem. For this reason, it has received significant attention in past research but most of these studies presented solutions in the centralized context.

In a distributed environment the view selection problem becomes more challenging. Indeed, it includes another issue which is to decide on which computer nodes the selected views should be materialized. Furthermore, resource constraints such as CPU, IO, network bandwidth have to be taken into consideration. The view selection problem in a distributed context may also be constrained by storage space capacities per computer node and maximum view maintenance cost. To the best of our knowledge, no past work has addressed this problem under all these

resource constraints. Our constraint programming based approach fills this gap. Indeed, all these resource constraints will easily be modeled with the rich constraint programming language. Furthermore, the heuristic algorithms which have been designed to solve the view selection problem in a distributed scenario are deterministic algorithms. For example greedy algorithm [3] and genetic algorithm [4], a type of randomized algorithms. These heuristic algorithms may provide near optimal solutions but there is no guarantee to find the global optimum because of their greedy nature or their probabilistic behavior. We have demonstrated in our recent work [10] the benefit of using constraint programming techniques for solving the view selection problem with reference to the centralized context in terms of the solution quality. Indeed, our approach 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 until reaching the optimal solution. Specifically, our main contributions are:

1. We propose an extension of the concept of the AND-OR view graph [14] in order to reflect the relation between views and communication network within the distributed scenario. We make use of the concept of the AND-OR view graph to exhibit common sub-expressions between queries of workload which can be exploited for sharing updates and storage space.
2. We describe how to model the view selection problem in a distributed context as a Constraint Satisfaction Problem (CSP). Its resolution is supported automatically by the constraint solver such as the powerful version of CHOCO [1]. The view selection problem has been addressed under multiple resource constraints. The limited resources are the total view maintenance cost and the storage space capacity for each computer node. Furthermore, we consider the IO and CPU costs for each computer node as well as the network bandwidth.
3. We have implemented our approach and compared it with a randomized method i.e., genetic algorithm [4] which has been designed for a distributed setting. We experimentally show that our approach provides better performance resulting from evaluating the quality of the solutions in terms of cost saving.

The rest of this paper is organized as follows. Section 2 presents the framework that we have designed specifically to a distributed setting. Section 3 describes how to model the view selection problem under multiple resource constraints in a distributed environment as a constraint satisfaction problem. In section 4, it is provided our experimental evaluation. Section 5 presents a brief survey of related work. Finally, section 6 contains concluding remarks and future work.

## 2 The Distributed AND-OR View Graph

In order to exhibit common sub-expressions between queries of workload, the view selection is represented by using a AND-OR view graph [14]. Common sub-expressions can be exploited for sharing updates and storage space. The AND-OR view graph is a Directed Acyclic Graph (DAG) which is 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 Eq-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 and the leaf nodes represent the base relations. Equivalence nodes correspond to the views that are candidates to materialization.

The AND-OR view graph is the union of all possible execution plans of each query. Our motivation to consider all execution strategies is that it has been argued that a good selection of materialized views can only be found by considering the optimization of both global processing plans and materialized view selection [16]. The AND-OR view graph of the queries  $q_1 = P \text{ join } PS \text{ join } S$  and  $q_2 = PS \text{ join } S \text{ join } N$  where P, PS, S and N are the base relations <sup>1</sup> is shown in

---

<sup>1</sup>The subscripts P, PS, S and N denote respectively the base relations of TPC-H benchmark: Part, PartSupp, Supplier and Nation

figure 1. Circles represent operation nodes and boxes represent equivalence nodes. For simplicity, we represent only two execution plans for the query  $q_1$  and one execution plan for the query  $q_2$ . The remaining execution plans are just indicated by dashed lines. For example, view P-PS-S, corresponding to query  $q_1$ , can be computed from P-PS and S or P and PS-S. This dependence is indicated in figure 1 by AND and OR arcs.

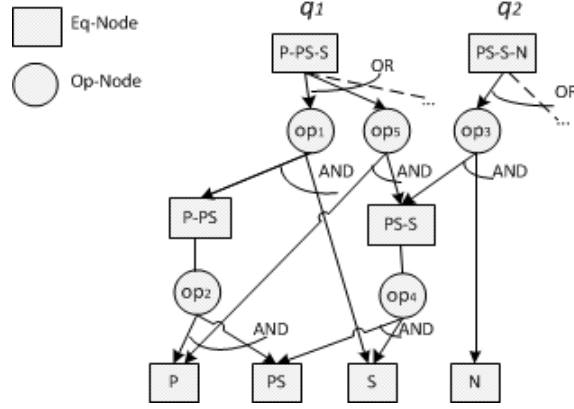


Figure 1: AND-OR view graph of two queries  $q_1$  and  $q_2$

In this paper, we extend the concept of the AND-OR view graph to deal with distributed settings. Therefore, we propose the distributed AND-OR view graph to reflect the relation between views and communication network in the distributed scenario. We consider a distributed setting involving a set of sites (computer nodes) with different resource constraints (CPU, IO, storage space capacity, network bandwidth), a set of queries, a set of updates and their respective frequencies. For each query  $q$ , we consider all possible execution plans which represent its execution strategies. In this paper we consider selection-projection-join (SPJ) queries that may involve aggregation and a group by clause as well. Let us consider the query  $q$  defined over a simplified version of the TPC-H benchmark [2]. Query  $q$  finds the minimal supply cost for each country and each product having the brand name 'Renault'. The associated query is as follows:

```

Select    P.partkey, N.nationkey, N.name, Min(PS.supplycost)
From     Part P, Supplier S, Nation N, PartSupp PS
Where    P.brand = 'Renault'
and      P.partkey = PS.partkey
and      PS.supkey = S. supkey
and      S.nationkey = N.nationkey
Group by P.partkey, N.nationkey, N.name;

```

A sample distributed AND-OR view graph is shown in figure 2. For simplicity, we consider a network of only three sites  $s_1, s_2, s_3$  and we illustrate a part of the query  $q$  by considering only join operations and one execution strategy. Indeed, in figure 2 we consider only the join between Part (P) and PartSupp (PS) and the join between PartSupp (PS) and Supplier (S). The execution strategy that we have presented in figure 2 is  $((P \text{ join } PS) \text{ join } S)$ . We suppose that the base relations are stored on different sites.

In order to represent the communication channels, every node is split into three sub-nodes, each of which denotes the view or the execution operation at one site. The communication edges between equivalence nodes of the same level (i.e.,  $(P - PS - S, S_1)$ ,  $(P - PS - S, S_2)$  and  $(P - PS - S, S_3)$ ), as shown in the dashed rectangle in figure 2, denote that a view can be answered from any other site if it is less expensive than computing this view from any children nodes. However, these edges are bidirectional creating cycles which no longer conforms to the characteristics of a DAG. In order to eliminate cycles, each sub-node  $(v_i, S_j)$ , as illustrated in figure 3, has been artificially split into two nodes  $(v_i, S_j)'$  and  $(v_i, S_j)''$ .

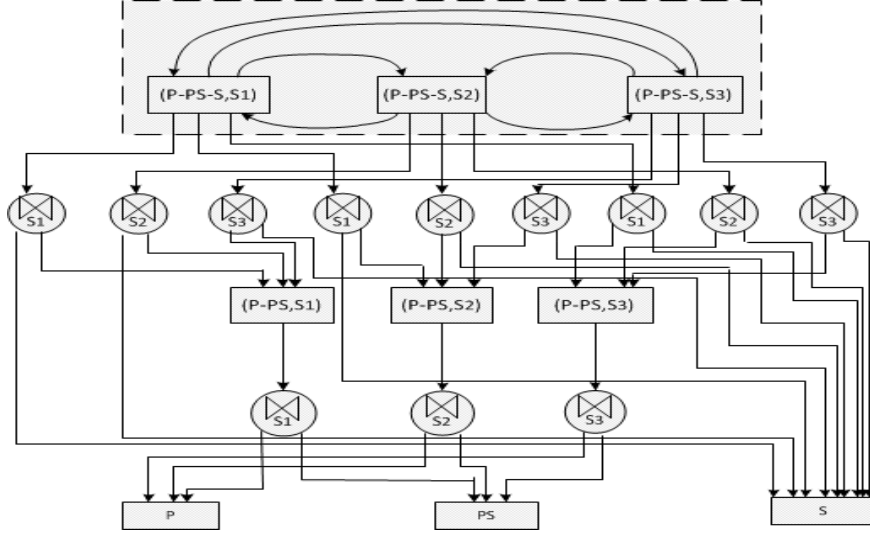


Figure 2: Distributed AND-OR view graph.

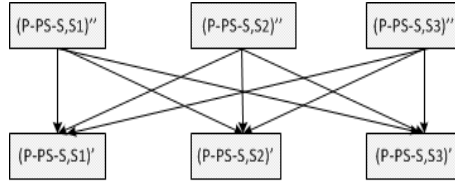


Figure 3: Modified Distributed AND-OR view graph.

### 3 Modeling View Selection Problem in a Distributed Context as a Constraint Satisfaction Problem (CSP)

In this subsection, we describe how to model the view selection problem in a distributed scenario as CSP. Then, its resolution is supported automatically by the constraint solver. In what follows, we define all the symbols as well as the variables that we have used in our CSP.

- $G$ . The distributed AND-OR view graph.
- $Q(G)$ . The query workload. \*
- $V(G)$ . The set of candidate views.
- $U$ . The set of updates.
- $\delta(v_i, s_j, u)$ . The differential result of view  $v_i$  on  $s_j$ , with respect to update  $u$ .
- $f_q$ . The frequency of a query.
- $f_u$ . The update frequency of a query (or view).
- $S$ . The set of sites which represent the computer nodes.
- $Sp_{max_i}$ . The storage space capacity of the site  $s_i$ .
- $U_{max}$ . The maximum view maintenance cost.
- $|v_i|$ . The size of  $v_i$  in terms of number of bytes.
- $Bw(s_k, s_j)$ . The bandwidth between  $s_j$  and  $s_k$ .

#### CSP variables and their domains

- $Mat(v_i, s_j)$ . The materialization of the view  $v_i$  on site  $s_j$ . It is a binary variable ( $d_{Mat(v_i, s_j)} = 0, 1$ ; 0:  $v_i$  is not materialized on  $s_j$ , 1:  $v_i$  is materialized on  $s_j$ ).
- $Qc(v_i, s_j)$ . The query cost corresponding to the view  $v_i$  if it is computed or materialized on site  $s_j$ . The domain is a finite subset of  $\mathbb{R}_+^*$  ( $d_{Qc(v_i, s_j)} \subset \mathbb{R}_+^*$ ).
- $Mc(v_i, s_j)$ . The maintenance cost corresponding to the view  $v_i$  if it is updated on site  $s_j$ , where  $d_{Mc(v_i, s_j)} \subset \mathbb{R}_+^*$ .

The view selection in a distributed scenario can be formulated by the following constraint satisfaction model.

$$\text{minimize} \quad \sum_{(v_i, s_j) \in Q(G)} \left( f_q(v_i) * Qc(v_i, s_j) \right) \quad (1)$$

$$\text{subject to} \quad \forall s_j \in S \quad \sum_{(v_i, s_j) \in V(G)} \left( Mat(v_i, s_j) * |v_i| * IO_j \right) \leq Sp_{max_j} \quad (2)$$

$$\sum_{(v_i, s_j) \in V(G)} \left( Mat(v_i, s_j) * f_u(v_i) * Mc(v_i, s_j) \right) \leq U_{max} \quad (3)$$

In our approach, the main objective is the minimization of the total query cost. The total query cost is computed by summing over the cost of processing each input query rewritten over the materialized views. Constraints (2) and (3) state that the views are selected to be materialized on a set of sites under a limited amount of resources. Constraint (2) ensures that for each site the total space occupied by the materialized views on it is less than its storage space capacity. Constraint (3) guarantees that the total maintenance cost of the set of materialized views is less than the maximum view maintenance cost.

The query and maintenance costs may be formulated as follows.

$$Qc(v_i, s_j) = \min_{s_k \in S} \left( Qc_{local}(v_i, s_k) + \frac{|v_i|}{Bw(s_k, s_j)} \right) \quad (4)$$

$$Qc_{local}(v_i, s_j) = \begin{cases} ComputingCost(v_i, s_j) & \text{if } Mat(v_i, s_j) = 0 \\ |v_i| * IO_j & \text{otherwise} \end{cases} \quad (5)$$

$$ComputingCost(v_i, s_j) = \min_{op_l \in child(v_i, s_j)} \left( cost(op_l, s_j) + \sum_{(v_m, s_n) \in child(op_l)} \left( Qc(v_m, s_n) + \frac{|v_m|}{Bw(s_n, s_j)} \right) \right) \quad (6)$$

**Query Cost.** The query cost includes the local processing cost and the communication cost. The local processing cost reflects CPU and IO costs. Constraint (4) guarantees that a view is answered from the site that can provide the answer with the lowest cost. Constraint (5) and (6) ensure that the minimum cost path is selected for computing a given view on a given site. 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. The reading cost is considered if the view has been materialized.

$$Mc(v_i, s_j) = \begin{cases} 0 & \text{if } Mat(v_i, s_j) = 0 \\ \sum_{u \in U(v_i, s_j)} \left( \min_{s_k \in S} \left( Mcost(v_i, s_k, u) + \frac{|v_i|}{Bw(s_k, s_j)} \right) \right) & \text{otherwise} \end{cases} \quad (7)$$

$$Mcost(v_i, s_j, u) = \min_{op_l \in child(v_i, s_j)} \left( cost(op_l, s_j, u) + \sum_{(v_m, s_n) \in child(op_l)} \left( UpdatingCost(v_m, s_n, u) + \frac{|v_m|}{Bw(s_n, s_j)} \right) \right) \quad (8)$$

$$UpdatingCost(v_m, s_n, u) = \begin{cases} Mcost(v_m, s_n, u) + \frac{|v_i|}{Bw(s_n, s_m)} & \text{if } Mat(v_m, s_n) = 0 \\ \delta(v_m, s_n, u) & \text{otherwise} \end{cases} \quad (9)$$

**View Maintenance Cost.** The view maintenance cost is computed by summing the number of changes in the base relations from which the view is updated. 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. Constraint (7) guarantees that a view with respect to the updates of the underlying base relations is updated from the site that can provide the differential results with the lowest cost. Constraints (8) and (9) insure that the best plan with the minimum cost is selected to maintain a view. 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 on the path and the maintenance cost corresponding to the related views.

## 4 Experimental Evaluation

In this section, we demonstrate the performance of our approach and compare it to the one of a randomized method i.e., genetic algorithm which has been designed for a distributed setting [4]. The performance of view selection methods was evaluated by measuring the solution quality which results from evaluating the quality of the obtained set of materialized views in terms of cost saving.

### 4.1 Experiment Settings

For our experiments, we implemented a simulated distributed environment including a network of a set of sites (computer nodes). We assume that the different sites are divided into clusters so that there is a high probability that the sites which belong to the same cluster have similar query workloads. In our approach, for each cluster all the queries of the different workloads are merged into the same graph in order to detect the overlapping and capture the dependencies among them. Then, our method decides which views have to be selected and determine where these views should be materialized so that the full query workload is answered with the lowest cost under multiple resource constraints. The query workload are defined over the database schema of the TPC-H benchmark [2]. We then randomly assigned values to the frequencies for access and update based on a uniform distribution. In order to solve the view selection problem in a distributed context as a constraint satisfaction problem, we have used the latest powerful version of CHOCO [1]. For the randomized method, we have implemented the genetic algorithm presented in [4] by incorporating space and maintenance cost constraints into the algorithm. 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 space and maintenance cost constraints are most likely selected for materialization. In the experimental results, the solution quality denoted by  $Q_s$  is computed as follows.

$$Q_s = 1 - \frac{\sum_{(v_i, s_j) \in Q(G)} (f_q(v_i) * Qc(v_i, s_j))}{WM} \quad (10)$$

Where  $WM$  is the total query cost obtained using the "WithoutMat" approach which does not materialize views and always recomputes queries. The "WithoutMat" approach is used as a

benchmark for our normalized results. Recall that  $Qc(v_i, s_j)$  is the query cost corresponding to the view  $v_i$  on site  $s_j$  and  $f_q(v_i)$  is the frequency of the view  $v_i$  corresponding to a single query.

In our approach, the view selection problem in a distributed environment is constrained by storage capacities  $Sp_{max} = \{Sp_{max_i}, Sp_{max_j}, \dots, Sp_{max_n}\}$  where each site  $s_i$  has an associated storage space capacity  $Sp_{max_i}$  and maximum view maintenance cost  $U_{max}$ . Similar to [7] the storage space and maintenance cost limits are computed respectively as a function of the size (see equation 11) and total maintenance cost (see equation 12) of the query workload.

$$Sp_{max_i} = \alpha * Sp_i(AllM) \quad (11)$$

$$U_{max} = \beta * Mc(AllM) \quad (12)$$

Where *AllM* is the "AllMat" approach which materializes the result of each query of the workload;  $\alpha$  and  $\beta$  are constant. In our experiments, the storage space limit is per site and computed as a function of the size of the associated query workload. The view maintenance cost limit is calculated as a function of the total maintenance cost when all the queries are materialized.

Our approach to solve the view selection problem in a distributed setting is able to provide optimal solutions. However, computing optimal solutions may be very expensive because of the great number of comparisons between all possible subsets of views which are candidate to materialization. In this case, we use *timeout* condition to limit the search by considering that some solutions should not be explored. Note that the constraint solver can find a set of feasible solutions in which all the constraints are satisfied before reaching the optimal solution. In the next experiments, the constraint solver performed a search until reaching the *timeout* condition. Indeed, our approach is able to provide a feasible solution at any time. The *timeout* condition was set to the time required by the genetic algorithm to solve the problem. This means that the constraint solver was left to run until the convergence of the genetic algorithm in the following experiments.

## 4.2 Experiment Results

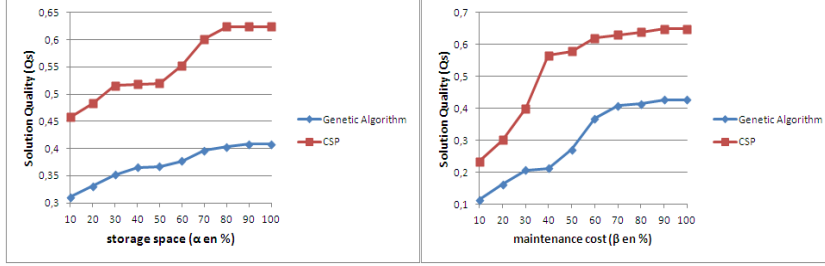
We examined the effectiveness of our approach within three experiments. The first one compares the performance of our approach and the genetic algorithm for various values of storage space and maintenance cost limits. The second experiment evaluates the view selection methods with respect to different sizes of the distributed AND-OR view graph in terms of number of views (equivalence nodes). Finally, the last experiment evaluates our approach and the genetic algorithm with different network sizes in terms of the number of sites per cluster.

### 4.2.1 Performances under resource constraints.

In this experiment, we examine the impact of space and maintenance cost constraints on solution quality. For this evaluation, each cluster includes 8 sites with different constraints of CPU, IO and network bandwidth and each site has an associated query workload. The values of  $\alpha$  and  $\beta$  which define respectively the storage space capacities and the view maintenance cost limit are varied from 10% to 100%. All the results are shown in figure 4. Figure 4 (a) investigates the influence of space constraint on solution quality for each value of  $\alpha$  where  $\beta$  was set to 60%. We note that the quality of the solutions produced by our approach and genetic algorithm improves when  $\alpha$  increases, since there is storage space available for more views to be materialized. However, when  $\alpha \geq 80\%$  there is no improvement in the solution quality because the maintenance cost constraint becomes the significant factor. Figure 4 (b) examines the impact of maintenance cost constraint on solution quality for each value of  $\beta$  where  $\alpha$  was set to 80%. We can observe similarly to figure 4 (a) that we have better solutions when  $\beta$  increases since there is time to update the materialized views. The performance stabilizes when  $\beta \geq 90\%$  because the space constraint becomes the significant factor. We note from these experiments that our approach outperforms the genetic algorithm in



the case where the resource constraints become very tight as well as in the case where we relax them. Indeed, for different values of  $\alpha$  and  $\beta$  we can see that our approach generates solutions with cost saving more than 2 times more than the genetic algorithm.

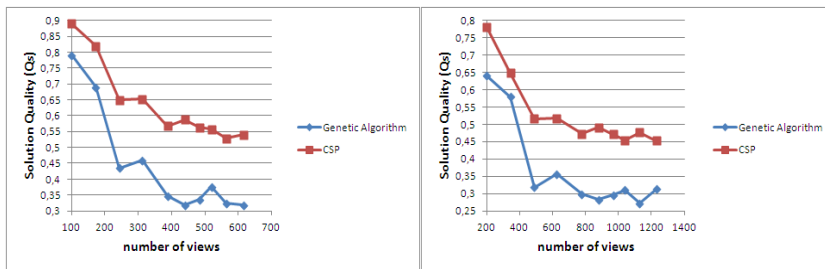


(a) Solution quality while varying the space constraint (b) Solution quality while varying the maintenance cost constraint

Figure 4: Evaluating the performance under resource constraints.

#### 4.2.2 Performance according to the number of views.

Let us now evaluate the performance of our approach and the one of genetic algorithm while varying the size of the search space. Recall that the size of the search space is estimated according to the number of views (equivalence nodes) in the distributed AND-OR view graph described in section 3. Figure 5 illustrates the quality of the solutions produced by the two methods in a distributed environment. The number of sites per cluster is 4 sites in figure 5 (a) and 8 sites in figure 5 (b). The queries of the workload are randomly distributed over the network so that each site has an associated query workload. For instance, in figure 5 (b), the number of views in the distributed AND-OR view graph ranges from 200 to 1232 views. For each site,  $\alpha$  was set to 40%. For the maintenance cost constraint,  $\beta$  was set to 60%. The experiment results depicted in figure 5 (a) and 5 (b) show that our approach provides the lowest query cost while varying the number of views. In fact, the cost saving is up to 27% more than the genetic algorithm. Therefore, our approach provides better performances compared with the genetic algorithm in terms of the solution quality.



(a) Number of sites=4 (b) Number of sites=8

Figure 5: Evaluating the performance over different number of views.

#### 4.2.3 Performance according to the number of sites.

In order to evaluate the performance of view selection methods according to the number of sites, we conducted experiments with clusters of different sizes. For each cluster, we considered different number of sites with different constraints of CPU, IO and network bandwidth. The number of sites per cluster varies from 2 to 20. For each site,  $\alpha$  was set to 40% and for the maintenance cost constraint,  $\beta$  was set to 60%. The experiment results are shown in figure 6. As in the previous

experiments, we observe that our approach provides an improvement in the quality of the obtained set of materialized views in terms of cost saving compared with the genetic algorithm. Indeed, the cost saving is up to 15% more than the genetic algorithm.

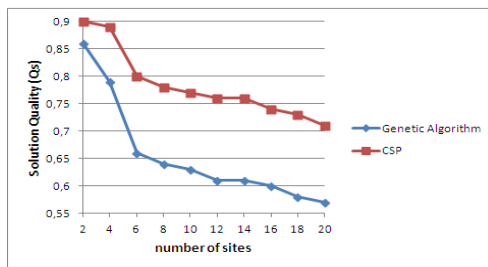


Figure 6: Evaluating the performance over different number of sites.

## 5 Related Work

Several view selection methods have been proposed in the literature to select which views to materialize in a centralized context. They can be classified into four major groups.

*Deterministic methods:* Methods in this class take a deterministic approach by exhaustive search [8, 13] or by some heuristics such as greedy [5, 6, 12, 14, 15]. However, greedy search is subjected to the known caveats, i.e., sub-optimal solutions may be retained instead of the globally optimal one since initial solutions influence the solution greatly.

*Randomized methods:* Typical algorithms in the context of view selection are genetic [9, 17] or use simulated annealing [7]. Randomized algorithms can be applied to complex problems dealing with large or even unlimited search spaces. However, the quality of the solution depends 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. Furthermore, randomized algorithms do not guarantee to find the global optimum because of their probabilistic behavior.

*Hybrid methods:* Hybrid methods combine the strategies of deterministic and randomized algorithms in their search. A hybrid approach has been applied in [16] to the view selection problem which combine heuristic algorithms i.e., greedy algorithms and genetic algorithms. They prove that hybrid algorithms provide better solution quality. However, they are more time consuming and may be impractical due to their excessive computation time.

*Constraint Programming methods:* A constraint programming based approach has been presented in our previous work [10] to address the view selection problem in a centralized context. We have proved experimentally that our approach provides better performance compared with a randomized method i.e., genetic algorithm in term of cost savings. The success of using constraint programming for combinatorial optimization is due to its combination of high level modeling, constraint propagation and facilities to control the search behavior.

Analysis of view selection methods has shown that there is little work on view selection in a distributed scenario. The view selection problem is addressed in a distributed data warehouse environment in [3]. An extension of the concept of a data cube lattice to capture the distributed semantics has been proposed. Moreover, they extend a greedy based selection algorithm to the distributed case. However, the cost model that they have used does not include the view maintenance cost. Furthermore, the network transmission costs are not considered which is very important in a distributed context. The study presented in [4] deals with the view selection problem in distributed databases. This approach consists in applying a genetic algorithm to select a set of materialized views and the nodes of the network on which they will be materialized. However, this approach does not take into account neither the space nor the maintenance cost constraint. Besides, our approach provides better results compared with genetic algorithm in terms of the solution quality. A survey of view selection methods can be found in our previous work [11].

## 6 Conclusion

In this paper we have designed a constraint programming based approach to address the view selection problem under multiple resource constraints in a distributed environment. Furthermore, we have introduced the distributed AND-OR view graph to reflect the relation between views and communication network. We have performed several experiments over TPC-H queries and comparison with a genetic algorithm. The experiment results have shown that our approach provides better performance where the space and maintenance cost constraints become very tight as well as in the case where we relax them or when the number of views is high. Besides, our approach provides better solution quality in terms of cost saving when we consider diverse number of sites.

As a future work, we plan to design a set of pruning heuristics in order to reduce the search space of candidate views to materialization. This means that the size of the distributed AND-OR view graph will be small enough to allow its use for solving the view selection problem in a large scale distributed environment within reasonable execution time. The design of these heuristics will also guarantee the optimality of the solution where no time limit is imposed.

## References

- [1] Choco, open-source software for constraint satisfaction problems. <http://www.emn.fr/z-info/choco-solver>.
- [2] The TPC benchmark H (TPC-H). <http://www.tpc.org/tpch/spec/tpch2.14.3.pdf>.
- [3] A. Bauer and W. Lehner. On solving the view selection problem in distributed data warehouse architectures. In *SSDBM*, pages 43–, 2003.
- [4] L.W.F. Chaves, E. Buchmann, F. Hueske, and K. Böhm. Towards materialized view selection for distributed databases. In *EDBT*, pages 1088–1099, New York, NY, USA, 2009. ACM.
- [5] H. Gupta. Selection of views to materialize in a data warehouse. In *ICDT*, pages 98–112, 1997.
- [6] Himanshu Gupta and Inderpal Singh Mumick. Selection of views to materialize under a maintenance cost constraint. In *ICDT*, pages 453–470, 1999.
- [7] P. Kalnis, N. Mamoulis, and D. Papadias. View selection using randomized search. *Data Knowl. Eng.*, 42(1):89–111, 2002.
- [8] Wilburt Labio, Dallan Quass, and Brad Adelberg. Physical database design for data warehouses. In *ICDE*, pages 277–288, Washington, DC, USA, 1997. IEEE Computer Society.
- [9] 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.
- [10] I. Mami, R. Coletta, and Z. Bellahsene. Modeling view selection as a constraint satisfaction problem. In *DEXA (2)*, pages 396–410, 2011.
- [11] Imene Mami and Zohra Bellahsene. A survey of view selection methods. *SIGMOD Record*, 41(1):20–29, 2012.
- [12] H. Mistry, P. Roy, S. Sudarshan, and K. Ramamritham. Materialized view selection and maintenance using multi-query optimization. In *SIGMOD Conference*, pages 307–318, 2001.
- [13] K.A. Ross, D. Srivastava, and S. Sudarshan. Materialized view maintenance and integrity constraint checking: Trading space for time. In *SIGMOD Conference*, pages 447–458, 1996.
- [14] P. Roy, S. Seshadri, S. Sudarshan, and S. Bhohe. Efficient and extensible algorithms for multi query optimization. In *SIGMOD Conference*, pages 249–260, 2000.
- [15] Jian Yang, Kamalakar Karlapalem, and Qing Li. Algorithms for materialized view design in data warehousing environment. In *VLDB*, pages 136–145, 1997.
- [16] C. Zhang, X. Yao, and J. 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.
- [17] Chuan Zhang and Jian Yang. Genetic algorithm for materialized view selection in data warehouse environments. In *DaWaK*, pages 116–125, 1999.