A Survey of View Selection Methods

Imene Mami and Zohra Bellahsene
Université Montpellier 2 - INRIA, LIRMM, Montpellier, France
{imen.mami,bella@lirmm.fr}

ABSTRACT

Materialized view selection is a critical problem in many applications such as query processing, data warehousing, distributed and semantic web databases, etc. We refer to the problem of selecting an appropriate set of materialized views as the view selection problem. Many different view selection methods have been proposed in the literature to address this issue. The present paper provides a survey of view selection methods. It defines a framework for highlighting the view selection problem by identifying the main dimensions that are the basis in the classification of view selection methods. Based on this classification, this study reviews most of the view selection methods by identifying respective potentials and limits.

1. INTRODUCTION

The view selection issue has been investigated in several contexts: query optimization, warehouse design, data placement in a distributed setting, web databases, etc. Many diverse solutions to the view selection problem have been proposed and analyzed through surveys [13, 21, 32]. The survey [21] concentrates on methods of finding a rewriting of a query using a set of materialized views. The study presented in [32] focuses on the state of the art in materialization for web databases. A critical analysis of methodologies for selecting materialized views in data warehousing is provided in [13]. However, none of the above mentioned surveys provides a classification of view selection approaches in order to identify their advantages and disadvantages. Our survey fills this gap.

This paper aims at studying the view selection in relational databases and data warehouses as well as in a distributed setting. First, we define a framework for highlighting the view selection problem. Thus, we present a classification of view selection methods based on the main view selection dimensions that we have identified. This study also reviews existing view selection methods by identifying respective potentials and limits.

The rest of the paper is organized as follows: Section 2 gives some definitions. Section 3 identifies the main view selection dimensions along which view selection methods can be classified. Section 4 presents a critical survey of existing view selection methods. Section 5 contains the conclusion and discusses open issues.

2. PROBLEM SPECIFICATION

2.1 Preliminaries

Here, we introduce the main notions used in this paper and related to the view selection context.

View: A view is a derived relation, defined by a query in terms of base relations and/or other views.

Materialized View: A view is said to be materialized if its query result is persistently stored otherwise it is said to be virtual. We refer to a set of selected views to materialize as a set of materialized views.

Workload: A workload or a query workload is a given set of queries $Q = \{Q_1, Q_2, \cdots, Q_q\}$. Each query Q_i has an associated non-negative weight fQ_i which describes the query frequency. The set of materialized views is dependent on the query workload. In a distributed scenario, the queries are executed on different computer nodes. Each computer node has an associated query workload.

View Selection: Given a database schema and a query workload, the objective is to select an appropriate set of materialized views to improve query performance. The process of selecting a set of materialized views is known as view selection.

View Maintenance: 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 a materialized view is known as view maintenance. Different maintenance policies (deferred or immediate) and maintenance strategies (incremental or rematerialization)

can be applied [17, 18, 38, 58].

2.2 Problem Definition

The use of materialized views is a common technique to reduce query response time [6]. Indeed, materializing an appropriate set of views and answering queries using these views can significantly speed up the query processing since the access to materialized views can be much faster than recomputing the views. Therefore, materializing all the input queries can achieve the lowest query processing cost but the highest view maintenance cost since materialized views have to be maintained in order to keep them consistent with the data at sources. 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 parameters: query processing cost, view maintenance cost and storage space. The problem of choosing which views to materialize which have a desirable balance among the three costs is known as the view selection problem. This is one of the most challenging problems in data warehousing [50] and it is known to be a NP-complete problem [26]. In a distributed environment consisting of many heterogeneous nodes with different resource constraints, the distributed view selection problem is that which view has to be materialized at which node of the network. The view selection problem in a distributed case is much more difficult than the view selection problem in a central case because of the immense challenges associated to distributed settings [16] (i.e., data granularity, degrees of replication, heterogeneity of information sources, etc.).

Problem Formulation: The problem of view selection can be formulated as follows. Given a database schema $R = \{R_1, R_2, \cdots, R_r\}$, a query workload $Q = \{Q_1, Q_2, \cdots, Q_q\}$ defined over R, the problem is to select an appropriate set of materialized views $M = \{V_1, V_2, \cdots, V_m\}$ such that the query workload is answered with the lowest cost under a limited amount of resources, e.g., storage space and/or view maintenance cost.

The view selection problem in a distributed context consisting of a set of nodes $N = \{N_1, N_2, \cdots, N_n\}$ in which each node has an associated query workload, is to choose a set of views $M = \{V_1, V_2, \cdots, V_m\}$ and a set of nodes $N_v \subseteq N$ at which M should be materialized. The distributed view selection is designed so that the full query workload is answered with the lowest cost subject to resource constraints. Resources may be storage space per node, view maintenance cost, communication costs or a combination of them.

2.3 Cost Model

The cost model is an important issue for the view selection process [9]. The main objective in view selection problem is the minimization of the weighted query processing cost, defined by the formula:

$$QueryProcessingCost = \sum_{Q_i \in Q} f_{Q_i} * Qc(Q_i, M)$$

where f_{Q_i} is the query frequency of the query Q_i and $Qc(Q_i, M)$ is the processing cost corresponding to Q_i given a set of materialized views M.

Because materialized views have to be kept up to date, the view maintenance cost has to be considered. This cost is weighted by the update frequency indicating the frequency of updating materialized views. The view maintenance cost is computed as follows:

$$ViewMaintenanceCost = \sum_{V_i \in M} f_u(V_i) * Mc(V_i, M)$$

where $f_u(V_i)$ is the update frequency of the view V_i and $Mc(V_i, M)$ is the maintenance cost of V_i given a set of materialized views M.

The cost model is extended for distributed setting by taking into account the communication cost which is the cost for transferring data from its origin to the node that initiated the query. Given a query Q_i which is asked at a node N_j and denoting by V_k a view used to answer Q_i, the communication cost is zero if V_k is materialized at N_j. Otherwise, let N_l be the node containing V_k, then the communication cost for transferring V_k from N_l to N_j is:

$$CommunicationCost_{(V_k, N_l \rightarrow N_j)} = C_{N_j, N_l} * size(V_k)$$

where C_{N_j, N_l} is the network transmission cost per unit of data transferred between N_j and N_l and $size(V_k)$ is the size of the view V_k.

2.4 Static View Selection vs. Dynamic View Selection

A static view selection approach is based on a given workload and chooses accordingly the set of views to materialize. Whereas, in a dynamic view selection approach, the view selection is applied as a query arrives. Therefore, the workload is built incrementally and changes over time. Because the view selection has to be in synchronization with the workload, any change to the workload should be reflected to the view selection as well. Indeed, in a system of a dynamic nature [4, 5, 29], the set of

materialized views can be changed over time and replaced with more beneficial views in case of changing the query workload. In order to reduce view maintenance cost and storage space requirements, [57] aims at materializing the most frequently accessed tuples of the view rather than materializing all tuples of the view. The set of materialized tuples can be changed dynamically as the queries change, either manually or automatically by an internal cache manager using a feedback loop. However, the task of monitoring constantly the query pattern and periodically recalibrating the materialized views is rather complicated and time consuming especially in large data warehouse where many users with different profiles submit their queries.

A dynamic view selection is often referred to as view caching. With caching, the cache is initially empty and data are inserted or deleted from the cache during the query processing. Materialization could be performed even if no queries have been processed and materialized views have to be updated in response of changes on the base relations. A detailed comparison of these two techniques is given in [27]. Traditional caching approaches aim at caching the results of queries, in other words to cache views. Another alternative is to cache only a part of a view. Indeed, a chunk based scheme has been introduced in [12] for fine granularity caching. Chunk based caching allows caching of only few, frequently used tuples of views. To facilitate the computation of chunks required by a query but not found in the cache, a new organization for base relations has been proposed which they called a chunked file. Caching has been adopted in data warehousing [43], distributed databases [28] and peer to peer systems [25]. Dynamic view indexing has also been considered in [44]. In this paper, we focus only on static view selection methods because most of existing view selection approaches are of static nature.

3. VIEW SELECTION DIMENSIONS

In order to identify the advantages and disadvantages of view selection methods, we propose two main dimensions along with they can be classified: (i) Frameworks; and (ii) Resource Constraints.

3.1 Frameworks

Generally, approaches to the view selection problem consist of two main steps. The first step identifies the candidate views which are promising for materialization. Techniques based on multiquery DAG, syntactical analysis of the workload or query rewriting have been used to obtain the candidate views. Based on the set of candidate views, the second step selects the set of views to materialize under the resource constraints and by applying heuristic algorithms.

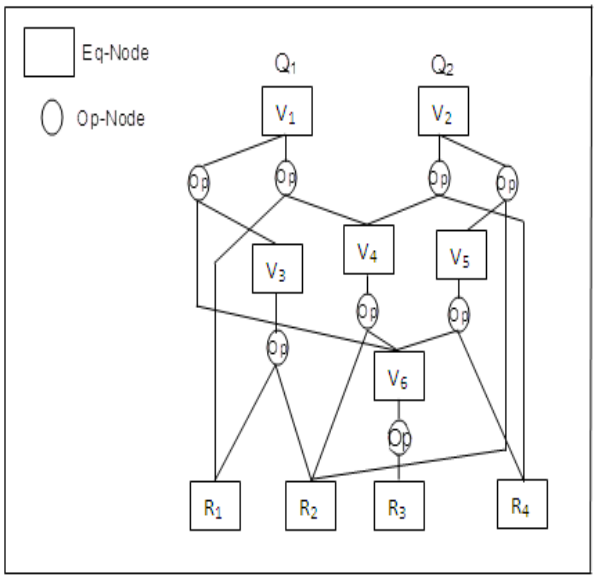

Figure 1: The AND-OR view graph of the two queries Q_1 and Q_2.

3.1.1 *Multiquery DAG*

Most of the proposed view selection methods operate on query execution plans. The plans can be derived from multiple query optimization techniques or by merging multiple query plans. The main interest of such techniques relies in detecting common sub-expressions between the different queries of workload and capturing the dependencies among them. This feature can be exploited for sharing updates and storage space. The dependence relation on queries (or views) has been represented by using a directed acyclic graph also called a DAG. However, these methods require optimizer calls which can be expensive in complex scenarios.

The most commonly used DAGs in literature are:

AND/OR View Graph: The union of all possible execution plans of each query forms an AND-OR view graph [40]. The AND-OR view graph described by Roy [42] is a Directed Acyclic Graph (DAG) composed of two types of nodes: Operation nodes and Equivalence nodes. Each operation node represents an algebraic expression (Select-Project-Join) with possible aggregate function. An equivalence node represents a set of logical expressions that are equivalent (i.e., that yield the same result). The operation nodes have only equivalence nodes as

children and equivalence nodes have only operation nodes as children. The root nodes are the query results and the leaf nodes represent the base relations. A sample AND-OR view graph is shown in figure 1. Circles represent operation nodes (Op-Nodes) and boxes represent equivalence nodes (Eq-Nodes). For example, in figure 1, view V_1 corresponding to a single query Q_1, can be computed from V_6 and V_3 or R_1 and V_4. If there is only one way to answer or update a given query, the graph becomes an AND view graph. In the data cube which is a specific model of a data warehouse, the AND-OR view graph is an OR view graph, as for each view there are zero or more ways to construct it from other views, but each way involves only one other view [19]. In other words, an OR view graph is an AND-OR view graph in which any node is an equivalence node that can be computed from any one of its children.

Multi-View Processing Plan (MVPP): The MVPP defined by Yang et al [52] is a directed acyclic graph in which the root nodes are the queries, the leaf nodes are the base relations and all other intermediate nodes are selection, projection, join or aggregation views that contribute to the construction of a given query. The MVPP is obtained after merging into a single plan either individual optimal query plans (similar to the AND view graph) or all possible plans for each query (similar to the AND-OR view graph). The difference between the MVPP representation and the AND-OR view graph or the AND view graph representation is that all intermediate nodes in the MVPP represent operation nodes. A sample MVPP is shown in figure 2.

Data Cube Lattice: Harinarayan and al [22] propose modeling data in multiple dimensions. It is built from the queries involved in the data warehouse application, e.g., OLAP-style queries. The Data Cube Lattice is a DAG whose nodes represent queries (or views) which are characterized by the attributes of the Group by clause. The edges denote the derivability relation between views. That is, if there is a path from view V_i to a view V_j (see figure 3), then grouping attributes on V_j can be calculated from grouping attributes on V_i. The node labeled none corresponds to an empty set of group-by attributes (tuples are not grouped). The benefit of this representation is that a query can be used to answer or update another query. An extension of the data cube lattice in order to adapt it to a distributed case was proposed in [3, 53]. Indeed, the cube has been modified by adding edges that mark the derivation relationship between views on different computer nodes.

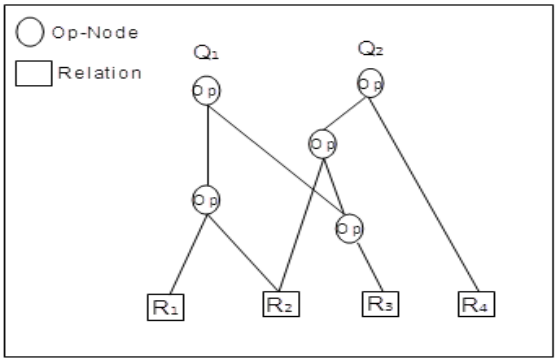

Figure 2: The MVPP of the two queries Q_1 and Q_2.

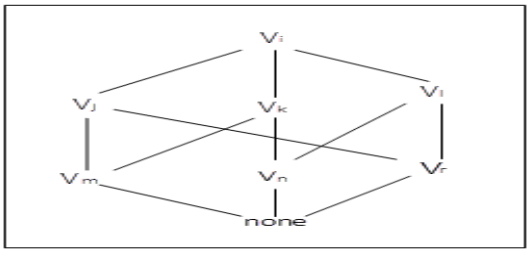

Figure 3: Sample Data Cube Lattice for eight views.

3.1.2 Query Rewriting

Query rewriting based approaches not only compute the set of materialized views but also find a complete rewriting of the queries over it. Here, the input to the view selection is not a multiquery DAG but the query definitions. The view selection problem is modeled as a state search problem using a set of transformation rules. These rules detect and exploit common subexpressions between the queries of the workload and guarantee that all the queries can be answered using exclusively the materialized views. Query rewriting based approaches not only compute the set of materialized views but also find a complete rewriting of the queries over it. Nevertheless, the completeness of the transformation rules may make the complexity of state space search strategies exponential.

3.1.3 Syntactical Analysis of the Workload

Some view selection methods are based on syntactical analysis of the workload to identify candidate views. These approaches analyze the workload and pick a subset of relations from which to materialize one or more views, if only if has the potential to reduce the cost of the workload significantly. How-

ever, the search space for computing the optimal set of views to be materialized may be very large.

3.2 Resource Constraints

Resource constraints considered during the view selection can be taken into account when classifying view selection methods. There are three main models presented in literature: unbounded, space constrained and maintenance cost constrained.

3.2.1 Unbounded

In the unbounded setting, there is no limit on available resources (storage, computation etc.). Thus, the view selection problem consists in choosing a set of views to materialize that minimizes the query processing cost and the view maintenance cost. Formally thus, the problem is:

$$minimize \left(\sum_{Q_i \in Q} f_{Q_i} * Qc(Q_i, M) + \sum_{V_i \in M} f_u(V_i) * Mc(V_i, M)\right)$$

However, this approach may lead to two kinds of problems. First, sometimes the selected views may be too large to fit in the available space. Second, the cost of the view maintenance may offset the performance advantages provided by the view materialization.

3.2.2 Space Constrained

Due to the storage space limitation, materializing all views is not always possible. In this setting, a useful notion is that of a view benefit (or query benefit). This is defined as the reduction in the workload evaluation cost, that can be achieved by materializing this view. Also relevant in this context is the *per-unit benefit*, obtained by dividing the view benefit by its space occupancy. It has been shown [19] that the per-space unit benefit of a view can only decrease as more views are selected (monotonic property). The space constrained model minimizes the query processing cost plus the view maintenance cost under a space constraint.

$$minimize \left(\sum_{Q_i \in Q} f_{Q_i} * Qc(Q_i, M) + \sum_{V_i \in M} f_u(V_i) * Mc(V_i, M)\right)$$

$$\text{under } \sum_{V_i \in M} size(V_i) \leq S$$

where S is the storage space capacity.

3.2.3 Maintenance Cost Constrained

This model constrains the time that can be allotted to keep up to date the materialized views in response to updates on base relations. In the maintenance cost constrained model, the maintenance cost of a view may decrease with selection of other views for materialization. Therefore, the query benefit per unit of maintenance cost of a view can increase [20]. This non monotonic nature of maintenance cost makes the view selection problem more difficult. The maintenance cost constrained model minimizes the query processing cost under a maintenance cost constraint.

$$minimize \left(\sum_{Q_i \in Q} f_{Q_i} * Qc(Q_i, M)\right)$$

$$\text{under } \sum_{V_i \in M} f_u(V_i) * Mc(V_i, M) \leq U$$

where U is the view maintenance cost limit.

The models that we have presented in section 3.2 can be extended to the distributed setting by taking into account the distributed specific features (i.e., the communication cost between the computer nodes).

4. REVIEW OF VIEW SELECTION METHODS

In this section, we classify the view selection methods according to several dimensions characterizing their algorithms (i) resource constraints they consider during the view selection process and (ii) frameworks they use to obtain the candidate views (see figure 4). Based on this classification, we review most of the view selection methods. The best-known heuristic algorithms proposed in litterature to solve the view selection problem, namely: deterministic algorithms, randomized algorithms, hybrid algorithms or constraint programming.

4.1 Deterministic Algorithms Based Methods

Much research work on view selection uses deterministic strategies to address the view selection problem. [41] is the first paper that provides a solution for materializing view indexes which can be seen as a special case of the materialized views. The solution is based on A* algorithm [37]. An exhaustive approach is also presented in [31, 39] for finding the best set of views to materialize. Nevertheless, an exhaustive search cannot compute the optimal solution in a reasonable time.

The authors in [22] present and analyze algorithms for view selection in case of OLAP-style queries. They provide a polynomial-time greedy algorithm to select a set of views to materialize that minimizes the query processing cost subject to a space constraint. However, this approach does not con-

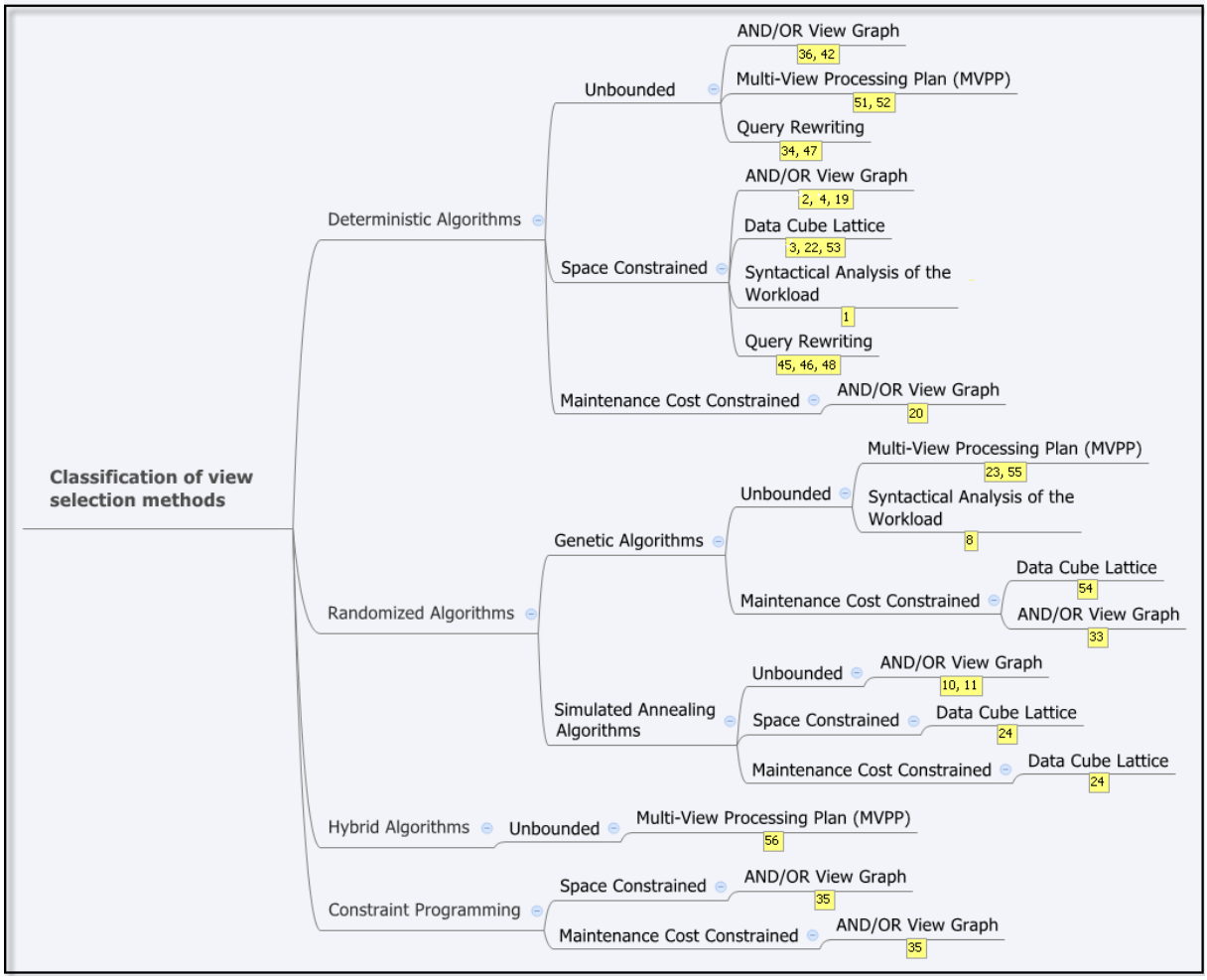

Figure 4: A Classification of view selection methods.

sider the view maintenance cost. The work in [51] is dealing with more general SQL queries which include select, project, join, and aggregation operations. A greedy algorithm has been designed to select a set of materialized views so that the combined query processing and view maintenance cost is minimized. However, the view maintenance cost has been overrated since the maintenance cost for a materialized view is the cost used for constructing this view. Besides, the view selection is done without any resource constraint.

A theoretical framework for the view selection problem in data warehousing setting has been developed in [19]. Their work provides a near-optimal exponential time greedy algorithm for the case of AND-OR view graph and near-optimal polynomial time greedy algorithm for the cases of AND view graph and OR view graph. This approach was extended in [20] to study the view selection under a maintenance cost constraint.

The authors in [42] demonstrate that using multi-query optimization techniques in conjunction with a greedy heuristic is practical and provides significant benefit. The greedy heuristic is used to iteratively pick from the AND-OR view graph the set of views to materialize that minimizes the query processing cost. This study was extended in [36] to consider how to optimize view maintenance cost. In addition to speed up the query workload by selecting materialized views, algorithms exploit common sub-expressions between view maintenance expressions to compute an efficient plan to the maintenance of the materialized views. However, the view selection has been studied without any resource constraint.

The view selection algorithm proposed in [2] is based on the notion of level in the query tree (each view of the query tree is associated to a level). In this approach, the view selection problem is studied under a space constraint and solved in two phases. The first phase depends on local optimization by

taking each query and pre-selecting a set of views which reduce the query processing cost without increasing significantly the view maintenance cost. The second phase computes the cost for each level of the query graph and selects the one which has the minimal sum of query processing and view maintenance cost.

The view selection has been studied in [34, 45, 46, 47, 48] under the condition that the input queries can be answered using exclusively the materialized views. An exhaustive algorithm has been designed in [47] to select a set of materialized views while minimizing the combination of the query processing and view maintenance cost. This work was extended in [34] by developing greedy algorithms that expand only a small fraction of the states produced by the exhaustive algorithm. The view selection problem in [45, 46, 48] is addressed under a space constraint. However, their view selection algorithm is still in exponential time. A survey of work on answering queries using views can be found in [21].

The study in [1] is based on a syntactical analysis of the workload to address the problem of selecting both views and indexes to be materialized. This approach proceeds in three main steps. The first step analyses the workload and chooses subsets of base relations with a high impact on the query processing cost. Based on the base relations subsets, the second step identify syntactically relevant views and indexes that can potentially be materialized. In the third step, the system runs a greedy enumeration algorithm to pick a set of views and indexes to materialize based on the result of the second step by taking into account the space constraint. Nevertheless, this approach does not take into account the view maintenance cost.

The works published in [3, 53] address the view selection problem in a distributed data warehouse environment. 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 for 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. Indeed, the communication cost is computed only from the size of the query result.

The above methods take a deterministic approach either by exhaustive search or by some heuristics such as greedy. 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. As a result, many paradigms and programming techniques have been developed to improve the solutions of the view selection problem: randomized algorithms, hybrid algorithms and constraint programming which we describe in next subsection.

4.2 Randomized Algorithms Based Methods

Typical randomized algorithms are genetic [14] or use simulated annealing [30]. Genetic algorithms generate solutions using techniques inspired by the natural evolution process such as selection, mutation, and crossover. The search strategy for these algorithms is very similar to biological evolution. Genetic algorithms start with a random initial population and generate new populations by random crossover and mutation. The fittest individual found is the solution. The algorithms terminate as soon as there is no further improvement over a period.

A genetic algorithm has been used in [23, 55] in conjunction with the MVPP framework to solve the view selection problem. The materialized views have been selected according to their reduction in the combined query processing and view maintenance cost. However, because of the random characteristic of the genetic algorithm, some solutions can be infeasible. For example, in the maintenance cost constrained model, when a view is selected, the benefit will not only depend on the view itself but also on other views that are selected. One solution to this problem is to add a penalty value as part of the fitness function to ensure that infeasible solutions will be discarded. For instance, a penalty function has been applied in [33] which reduces the fitness each time the maintenance cost constraint is not satisfied. This approach minimizes the query processing cost given varying upper bounds on the view maintenance cost, assuming unlimited amount of storage space. In order to let the genetic algorithm converge faster, they represent the initial population as a favorable configuration based on external knowledge about the problem and its solution rather than a random sampling, i.e., the views with a high query frequency are most likely selected for materialization. However, the genetic algorithm may tend to get stuck at a poor local optimum fairly early. A solution was provided in [54] to avoid premature convergence and keep improving the solution by incorporating constraints into the algorithm through a stochastic ranking procedure where no penalty functions are used.

The study presented in [8] which is based on a syntactical analysis of the workload deals with the

distributed view selection problem. This approach consists of three main steps. The first one extends the base relations selection algorithm described in [1] for the distributed scenario. Based on the result of the first step and the similarity between queries, the second step generates the candidate views which are promising for materialization. In the third step a genetic algorithm is applied to select a set of materialized views and the nodes of the network on which they will be materialized that minimize the query processing and view maintenance cost. However, this approach does not take into account either the space constraint or the maintenance cost constraint.

The approaches proposed in [10, 11, 24] use simulated annealing algorithms to address the view selection problem. These algorithms are motivated by an analogy to annealing in solids. Simulated Annealing algorithms start with an initial configuration, generate new configurations by random walk along the different solutions of the solution space according to a cooling schedule and terminate as soon as no applicable ones exist or lose all the energy in the system.

Materialized views have been selected in [10] so that the combined query processing and view maintenance cost is minimized. The view selection problem is solved in [24] under the case where either the space constraint or the maintenance cost constraint is considered. Further, randomized search has been applied to solve two more issues. First, they considered the case where both space and maintenance constraints exist. Next they applied randomized search in the context of dynamic view selection.

In order to support the scalability when the number of views and queries become large, a new approach has been introduced in [11] using Parallel Simulated Annealing (PSA) for materialized view selection. By performing simulated annealing with multiple inputs over multiple compute nodes concurrently, PSA is able to improve the quality of obtained sets of materialized views. Moreover, PSA is able to perform view selection on MVPP having a much larger number of views, which reflects the real data warehousing environment. However, the view selection problem is solved without any bound neither on the storage space nor on the view maintenance cost.

Randomized algorithms can be applied to complex problems dealing with large or even unlimited search spaces. Thus, the use of randomized algorithms can be considered in solving large combinatorial problems such as the view selection problem. However, the quality of the solution [1] 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.

4.3 Hybrid Algorithms Based Methods

Hybrid algorithms combine the strategies of deterministic and randomized algorithms in their search in order to provide better performance in terms of solution quality. Solutions obtained by deterministic algorithms are used as initial configuration for simulated annealing algorithms or as initial population for genetic algorithms.

A hybrid approach has been applied in [56] which combines heuristic algorithms i.e., greedy algorithms and genetic algorithms to solve three related problems. The first one is to optimize queries. The second one is to choose the best global processing plan from multiple processing plans for each query. The third problem is to select materialized views from a given global processing plan. Their experimental results confirmed that hybrid algorithms provide better performance than either genetic algorithms or heuristic algorithms i.e., greedy algorithms used alone in terms of solution quality. However, their algorithms are more time consuming and may be impractical due to their excessive computation time.

4.4 Constraint Programming Based Methods

Constraint programming is a descendant of declarative programming. This programming technique has been exploited in many applications for solving combinatorial problems [49]. 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.

A constraint programming based approach has been presented in [35] to address the view selection problem. More specifically, the view selection problem has been modeled as a constraint satisfaction problem. Its resolution has been supported automatically by constraint solver embedded in the constraint programming language. The authors proved experimentally that a constraint programming based approach provides better performances compared with a randomized method i.e., genetic algorithm in term of cost savings. The view selection has been studied under the case where (i) only the maintenance cost constraint is considered and (ii) both

[1]The solution quality represents the quality of the set of materialized views found by the algorithm. For example, the solution quality may be measured in term of cost savings.

maintenance cost and space constraints exist. They have also shown that their approach is scalable.

5. CONCLUSION

This study provides a critical survey of different approaches in which the view selection has been studied in relational databases and data warehouses as well as in a distributed setting. We have defined formally the view selection problem and identified the main view selection dimensions along with view selection methods have been classified. Based on the classification, we have discussed most of view selection methods.

Analysis of state of the art of view selection has shown that there is very few work on view selection in distributed databases and data warehouses [3, 8, 53] and no effective solution for peer to peer systems. Indeed, [16] seems to be the only paper which deals with the view selection problem in peer to peer environment. In fact, it is provided a full definition of the problem but without providing any algorithm or detail on how to select an effective set of views to materialize and place them at appropriate peers. Thus, one of challenging directions of future work aims at addressing the view selection problem in a distributed setting. More recently, materialized view selection has been explored in semantic web databases [7, 15] in order to facilitate efficient processing of RDF queries and updates. However, they consider a static workload which contradicts the dynamic nature of the web. Indeed, any change to the workload should be reflected to the view selection as well. This issue will be the future aspect while studying the view selection in semantic web databases.

6. ACKNOWLEDGEMENTS

We would like to thank the reviewers for their valuable comments to improve this paper.

7. REFERENCES

[1] S. Agrawal, S. Chaudhuri, and V.R. Narasayya. Automated selection of materialized views and indexes in SQL databases. In *VLDB*, pages 496–505, 2000.

[2] X. Baril and Z. Bellahsene. Selection of materialized views: A cost-based approach. In *CAiSE*, pages 665–680, 2003.

[3] A. Bauer and W. Lehner. On solving the view selection problem in distributed data warehouse architectures. In *SSDBM*, pages 43–, 2003.

[4] Z. Bellahsene, R. Coenmans, and J. Tranier. Matérialisation de vues dans les entrepôts de données. une approche dynamique. *Ingénierie des Systèmes d'Information*, 11(6):33–53, 2006.

[5] Z. Bellahsene, M. Cart, and N. Kadi. A cooperative approach to view selection and placement in P2P systems. In *OTM*, pages 515–522, 2010.

[6] R.G. Bello, K. Dias, A. Downing, J. Feenan, J.L. Finnerty, W.D. Norcott, H. Sun, A. Witkowski, and M. Ziauddin. Materialized views in ORACLE. In *VLDB*, pages 659–664, 1998.

[7] R. Castillo, and U. Leser. Selecting materialized views for RDF data. In *ICWE Workshops*, 2010.

[8] 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.

[9] R. Chirkova, A.Y. Halevy, and D. Suciu. A formal perspective on the view selection problem. *VLDB J.*, 11(3):216–237, 2002.

[10] R. Derakhshan, F.K.H.A. Dehne, O. Korn, and B. Stantic. Simulated annealing for materialized view selection in data warehousing environment. In *Databases and Applications*, pages 89–94, 2006.

[11] R. Derakhshan, B. Stantic, O. Korn, and F.K.H.A .Dehne. Parallel simulated annealing for materialized view selection in data warehousing environments. In *ICA3PP*, pages 121–132, 2008.

[12] P. Deshpande, K. Ramasamy, A. Shukla, and J.F. Naughton. Caching multidimensional queries using chunks. In *SIGMOD Conference*, pages 259–270, 1998.

[13] C.A. Dhote, and M.S. Ali. Materialized View Selection in Data Warehousing: A Survey. In *Journal of Applied Sciences*, pages 401–414, 2009.

[14] D.E. Goldberg. Genetic Algorithms in Search Optimization and Machine Learning. Addison-Wesley, 1989.

[15] F. Goasdoue, K. Karanasos, J. Leblay, and I. Manolescu. View Selection in Semantic Web Databases. In *PVLDB*, pages 97–108, 2011.

[16] S.D. Gribble, A.Y. Halevy, Z.G. Ives, M. Rodrig, and D. Suciu. What can database do for peer-to-peer? In *WebDB*, pages 31–36, 2001.

[17] A. Gupta and I.S. Mumick. Maintenance of materialized views: Problems, techniques, and applications. *IEEE Data Eng. Bull.*, 18(2):3–18, 1995.

[18] A. Gupta, I.S. Mumick, and V.S. Subrahmanian. Maintaining views incrementally. In *SIGMOD Conference*, pages 157–166, 1993.

[19] H. Gupta. Selection of views to materialize in a data warehouse. In *ICDT*, pages 98–112, 1997.

[20] H. Gupta and I.S. Mumick. Selection of views to materialize under a maintenance cost constraint. In *ICDT*, pages 453–470, 1999.

[21] A.Y. Halevy. Answering queries using views: A survey. *VLDB J.*, 10(4):270–294, 2001.

[22] V. Harinarayan, A. Rajaraman, and J.D. Ullman. Implementing data cubes efficiently. In *SIGMOD Conference*, pages 205–216, 1996.

[23] J.T. Horng, Y.J. Chang, and B.J. Liu. Applying evolutionary algorithms to materialized view selection in a data warehouse. *Soft Comput.*, 7(8):574–581, 2003.

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

[25] P. Kalnis, W.S. Ng, B.C. Ooi, D. Papadias, and K.L. Tan. An adaptive peer-to-peer network for distributed caching of OLAP results. In *SIGMOD Conference*, pages 25–36, 2002.

[26] H.J. Karloff and M. Mihail. On the complexity of the view-selection problem. In *PODS*, pages 167–173, 1999.

[27] D. Kossmann. The state of the art in distributed query processing. *ACM Comput. Surv.*, 32(4):422–469, 2000.

[28] D. Kossmann, M.J. Franklin, and G. Drasch. Cache investment: integrating query optimization and distributed data placement. *ACM Trans. Database Syst.*, 517–558, 2000.

[29] Y. Kotidis and N. Roussopoulos. Dynamat: A dynamic view management system for data warehouses. In *SIGMOD Conference*, pages 371–382, 1999.

[30] P.J.M. Laarhoven and E.H.L. Aarts, editors. *Simulated annealing: theory and applications*. Kluwer Academic Publishers, Norwell, MA, USA, 1987.

[31] W. Labio, D. Quass, and B. Adelberg. Physical database design for data warehouses. In *ICDE*, pages 277–288, Washington, DC, USA, 1997. IEEE Computer Society.

[32] A. Labrinidis , Q. Luo , J. Xu, and W. Xue. Caching and Materialization for Web Databases. In *Foundations and Trends in Databases*, pages 169–266, 2009.

[33] M. Lee and J. Hammer. Speeding up materialized view selection in data warehouses using a randomized algorithm. *Int. J. Cooperative Inf. Syst.*, 10(3):327–353, 2001.

[34] S. Ligoudistianos, D. Theodoratos, and T.K. Sellis. Experimental evaluation of data warehouse configuration algorithms. In *DEXA Workshop*, pages 218–223, 1998.

[35] I. Mami, R. Coletta, and Z. Bellahsene. Modeling view selection as a constraint satisfaction problem. In *DEXA*, pages 396–410, 2011.

[36] 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.

[37] N.J. Nilsson. *Problem-Solving Methods in Artificial Intelligence*. McGraw-Hill Pub. Co., 1971.

[38] D. Quass, A. Gupta, I.S. Mumick, and J. Widom. Making views self-maintainable for data warehousing. In *PDIS*, pages 158–169, 1996.

[39] 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.

[40] N. Roussopoulos. The logical access path schema of a database. *IEEE Trans. Software Eng.*, 8(6):563–573, 1982.

[41] N. Roussopoulos. View indexing in relational databases. *ACM Trans. Database Syst.*, 7(2):258–290, 1982.

[42] P. Roy, S. Seshadri, S. Sudarshan, and S. Bhobe. Efficient and extensible algorithms for multi query optimization. In *SIGMOD Conference*, pages 249–260, 2000.

[43] P. Scheuermann, J. Shim, and R. Vingralek. Watchman : A data warehouse intelligent cache manager. In *VLDB*, pages 51–62, 1996.

[44] K. Schnaitter, S. Abiteboul, T. Milo, and N. Polyzotis. Colt: continuous on-line tuning. In *SIGMOD Conference*, pages 793–795, 2006.

[45] D. Theodoratos, S. Ligoudistianos, and T.K. Sellis. Designing the global data warehouse with SPJ views. In *CAiSE*, pages 180–194, 1999.

[46] D. Theodoratos, S. Ligoudistianos, and T.K. Sellis. View selection for designing the global data warehouse. *Data Knowl. Eng.*, 39(3):219–240, 2001.

[47] D. Theodoratos and T.K. Sellis. Data warehouse configuration. In *VLDB*, pages 126–135, 1997.

[48] D. Theodoratos and T.K. Sellis. Data warehouse schema and instance design. In *ER*, pages 363–376, 1998.

[49] M. Wallace. Practical applications of constraint programming. *Constraints*, 1:139–168, 1996. 10.1007/BF00143881.

[50] J. Widom. Research problems in data warehousing. In *CIKM*, pages 25–30, 1995.

[51] J. Yang, K. Karlapalem, and Q. Li. Algorithms for materialized view design in data warehousing environment. In *VLDB*, pages 136–145, 1997.

[52] J. Yang, K. Karlapalem, and Q. Li. A framework for designing materialized views in data warehousing environment. In *ICDCS*, 1997.

[53] W. Ye, N. Gu, G. Yang, and Z. Liu. Extended derivation cube based view materialization selection in distributed data warehouse. In *WAIM*, pages 245–256, 2005.

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

[55] C. Zhang and J. Yang. Genetic algorithm for materialized view selection in data warehouse environments. In *DaWaK*, pages 116–125, 1999.

[56] 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.

[57] J. Zhou, P. Larson, J. Goldstein, and L. Ding. Dynamic materialized views. In *ICDE*, pages 526–535, 2007.

[58] Y. Zhuge, H.G. Molina, J. Hammer, and J. Widom. View maintenance in a warehousing environment. In *SIGMOD Conference*, pages 316–327, 1995.